Regression verification of some classical algorithms

Milena Vujošević-Janičić, Filip Marić

June, 2018.

Contents

Introduction

This file contains a corpus of programs that implement some classical algorithms often encountered in introductory programming and algorithms courses. Programs are classified into three sections: 1. programs that do not contain loops, 2. programs that contain loops whose equivalence is checked using the uninterpreted

function method, 3. programs that contain loops whose equivalence is checked using the k-equivalence method.

Loop-free programs

Number of days

- day_number.c
- Write a function that returns number of days in a given month of a given year.

```
int leap(int year) {
  return (year % 4 == 0 && year % 100 != 0) || (year % 400) == 0;
}

int day_number1(int month, int year) {
  int numDays;
  switch(month) {
    // january, march, may, july, august, october, december
  case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    numDays = 31;
    break;
    // april, jun, september, november
  case 4: case 6: case 9: case 11:
    numDays = 30;
    break;
    // february
  case 2:
    numDays = leap(year) ? 29 : 28;
    break;
  }
  return numDays;
}

int day_number2(int month, int year) {
  int numDays = 0;
  // january, march, may, july, august, october, december
  if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 || m
    numDays = 31;
  // april, june, september, november
  else if (month == 4 || month == 6 || month == 9 || month == 11)
    numDays = 30;
  // february
  else if (month == 2)
```

```
    numDays = leap(year) ? 29 : 28;
  return numDays;
}

int day_number3(int month, int year) {
  int numDaysInMonth[13];
  numDaysInMonth[0] = 0; numDaysInMonth[1] = 31; numDaysInMonth[2] = 28;
  numDaysInMonth[3] = 31; numDaysInMonth[4] = 30; numDaysInMonth[5] = 31;
  numDaysInMonth[6] = 30; numDaysInMonth[7] = 31; numDaysInMonth[8] = 31;
  numDaysInMonth[9] = 30; numDaysInMonth[10] = 31; numDaysInMonth[11] = 30;
  numDaysInMonth[12] = 31;
//    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

  int numDays = numDaysInMonth[month];
  if (month == 2 && leap(year))
    numDays++;

  return numDays;
}
```

Vowels

- vowels.c
- Write a function that checks if a given character is a vowel.

```
int vowels1(int a) {
  int result;
  switch(a) {
  case 'a': case 'e': case 'i': case 'o': case 'u':
  case 'A': case 'E': case 'I': case 'O': case 'U':
    result = 1;
    break;
  default : result = 0;
  }
  return result;
}

int vowels2(int c) {
    // evaluates to 1 (true) if c is a lowercase vowel
    int isLowercaseVowel = (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');

    // evaluates to 1 (true) if c is an uppercase vowel
    int isUppercaseVowel = (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U');

    // evaluates to 1 (true) if either isLowercaseVowel or isUppercaseVowel is true
```

```
    return (isLowercaseVowel || isUppercaseVowel);
}
```

Linear equation

- linear_eq.c
- Write a function that solves a given linear equation.

```
enum num {
  NO,
  ONE,
  INF
};

int linear1(int a, int b, float *x) {
  enum num n = ONE;
  if (a != 0.0) {
    *x = -b / a;
  } else {
    *x = 0;
    if (b == 0.0)
      n = INF;
    else
      n = NO;
  }
  return n;
}

int linear2(int a, int b, float *x) {

  if (a != 0) {
    *x = -b / a;
    return ONE;
  }
  *x = 0;
  if (b == 0.0)
    return INF;
  else
    return NO;
}
```

Quadratic equation

- quadratic_eq.c

- Write a function that calculates the number of solutions of a given quadratic equation.

```c
#include <math.h>

enum num {NO, ONE, TWO, INF};

int quadratic1(int a, int b, int c, int* x1, int* x2) {
  enum num n = NO;
  int d = b*b - 4*a*c;
  if (a != 0 && d>0) {
    n = TWO;
    *x1 = (-b+sqrt(d))/(2*a);
    *x2 = (-b-sqrt(d))/(2*a);
  }
  if (a != 0 && d==0) {
    n = ONE;
    *x1 = (-b)/(2*a);
    *x2 = (-b)/(2*a);
  }
  if (a != 0 && d<0) {
    n = NO;
    *x1 = 0;
    *x2 = 0;
  }
  if (a == 0 && b == 0 && c == 0) {
    n = INF;
    *x1 = 0;
    *x2 = 0;
  }
  if (a == 0 && b == 0 && c != 0) {
    n = NO;
    *x1 = 0;
    *x2 = 0;
  }
  if (a == 0 && b != 0) {
    n = ONE;
    *x1 = -c/b;
    *x2 = 0;
  }
  return n;
}

int quadratic2(int a, int b, int c, int* x1, int* x2) {
  if (a == 0) {
    // linear equation
```

```
    *x2 = 0;
    if (b == 0) {
      *x1 = 0;
      return c == 0 ? INF : NO;
    } else {
      *x1 = -c/b;
      return ONE;
    }
  } else {
    // genuine quadratic equation
    int d = b * b - 4 * a * c;
    if (d < 0) {
      *x1 = *x2 = 0;
      return NO;
    } else if(d == 0) {       *
        x1 = *x2 = -b / (2*a);
      return ONE;
    } else {
      *x1 = (-b + sqrt(d)) / (2 * a);
      *x2 = (-b - sqrt(d)) / (2 * a);
      return TWO;
    }
  }
}
```

Quadrant

- quadrant.c
- Write a function that calculates the quadrant of a point given its coordinates.

```
int quadrant1(int x, int y) {
  if (x > 0) {
    if (y > 0) {
      return 1;
    } else if (y < 0) {
      return 4;
    } else {
      return 5;
    }
  } else if (x < 0) {
    if (y > 0) {
      return 2;
    } else if (y < 0) {
      return 3;
```

```
    } else {
      return 6;
    }
  } else {
    if (y > 0) {
      return 7;
    } else if (y < 0) {
      return 8;
    } else {
      return 9;
    }
  }
  return 0;
}

int quadrant2(int x, int y) {
  if (x > 0 && y > 0)
    return 1;
  if (x > 0 && y < 0)
    return 4;
  if (x > 0 && y == 0)
    return 5;
  if (x < 0 && y > 0)
    return 2;
  if (x < 0 && y < 0)
    return 3;
  if (x < 0 && y == 0)
    return 6;
  if (x == 0 && y > 0)
    return 7;
  if (x == 0 && y < 0)
    return 8;
  if (x == 0 && y == 0)
    return 9;
  return 0;
}
```

Type of triangle

- triangle.c
- Write a function that determines the type of a triangle, given its three side lengths.

```
enum triangle {
  DIFF,
```

```
  TWO,
  ALL
};

int triangle1(int a, int b, int c) {
  if ((a == b) && (a == c))
    return ALL;
  if ((a == b) || (a == c) || (b == c))
    return TWO;
  return DIFF;
}

int triangle2(int a, int b, int c) {
  if (a == b) {
    if (a == c)
      return ALL;
    else
      return TWO;
  } else if (a == c)
    return TWO;
  else if (b == c)
    return TWO;
  else
    return DIFF;
}
```

Lexicographic comparison of points

- better.c
- Two students compete in two subjects: mathematics and programming. The winner is the one who had better overall score (the sum of points in two subjects). If they have the total score equal, then the winner is the one who scored better in programming. Write a function that given the points of two students determines if the first one was better than the second one.

```
int bolji1(int matA, int progA, int matB, int progB) {
  int ukupnoA = matA + progA;
  int ukupnoB = matB + progB;
  if (ukupnoA > ukupnoB)
    return 1;
  if (ukupnoA < ukupnoB)
    return 0;
  return progA > progB;
}
```

```c
int bolji2(int matA, int progA, int matB, int progB) {
  int ukupnoA = matA + progA;
  int ukupnoB = matB + progB;
  return (ukupnoA > ukupnoB) ||
         (ukupnoA == ukupnoB && progA > progB);
}
```

Legal age

- legal_age.c
- A person can vote if he is 18 years old. Write a function that checks if a person can vote, given its date of birth.

```c
int legal_age1(int d1, int m1, int g1, int d2, int m2, int g2) {
  if ((g2 > g1 + 18) ||
      (g2 == g1 + 18 && m2 > m1) ||
      (g2 == g1 + 18 && m2 == m1 && d2 >= d1))
        return 1;
  else
      return 0;
}

int porediDatume(int d1, int m1, int g1,
          int d2, int m2, int g2) {
  if (g1 != g2)
    return g1 - g2;
  if (m1 != m2)
    return m1 - m2;
  return d1 - d2;
}

int legal_age2(int d1, int m1, int g1, int d2, int m2, int g2) {
  if (porediDatume(d2, m2, g2, d1, m1, g1 + 18) >= 0)
        return 1;
  else
      return 0;
}
```

Compare times

- time.c
- Two friends arrived at the airport. Given their two arrival times (in hours, minuts, and seconds) write a function that checks who came first.

```
int hms_to_S(int h, int m, int s) {
  return h*60*60 + m*60 + s;
}

int ko_je_pre1(int hPera, int mPera, int sPera, int hMika, int mMika, int sMika)
{
  // convert times to seconds
  int SPera = hms_to_S(hPera, mPera, sPera);
  int SMika = hms_to_S(hMika, mMika, sMika);

  if (SPera < SMika)
    return 1;
  else if (SMika < SPera)
    return -1;
  else
    return 0;
}

int pre(int h1, int m1, int s1,
     int h2, int m2, int s2) {
  if (h1 < h2)
    return 1;
  if (h1 > h2)
    return 0;
  if (m1 < m2)
    return 1;
  if (m1 > m2)
    return 0;
  return s1 < s2;
}

int ko_je_pre2(int hPera, int mPera, int sPera, int hMika, int mMika, int sMika)
{
  if (pre(hPera, mPera, sPera, hMika, mMika, sMika))
    return 1;
  else if (pre(hMika, mMika, sMika, hPera, mPera, sPera))
    return -1;
  else
    return 0;
}

int poredi(int h1, int m1, int s1,
       int h2, int m2, int s2) {
  if (h1 - h2 != 0)
    return h1 - h2;
  if (m1 - m2 != 0)
```

```
    return m1 - m2;
  return s1 - s2;
}

int ko_je_pre3(int hPera, int mPera, int sPera, int hMika, int mMika, int sMika)
{
  int p = poredi(hPera, mPera, sPera, hMika, mMika, sMika);
  if (p < 0)
    return 1;
  else if (p > 0)
    return -1;
  else
    return 0;
}
```

Max of three numbers

- max3.c
- Write a function that finds tha maximum among three given numbers.

```
int max3_1(int a, int b, int c) {
  int m = a;
  if(m<b) m=b;
  if(m<c) m=c;
  return m;
}

int max3_2(int a, int b, int c) {
  if(a>=b && a>=c) return a;
  if(b>=a && b>=c) return b;
  if(c>=a && c>=b) return c;
  return a;
}

int max3_3(int a, int b, int c) {
  if(a>=b) {
    if(b>=c) return a;
    else if(a>=c) return a;
    else return c;
  }
  else if(b>=c) return b;
  else return c;
}
```

Max of four numbers

- max3.c
- Write a function that finds tha maximum among four given numbers.

```
int max4_1(int a, int b, int c, int d) {
    int m = a;
    if(m<b) m=b;
    if(m<c) m=c;
    if(m<d) m=d;
    return m;
}

int max4_2(int a, int b, int c, int d) {
    if(a>=b && a>=c && a>=d) return a;
    if(b>=a && b>=c && b>=d) return b;
    if(c>=a && c>=b && c>=d) return c;
    if(d>=a && d>=b && d>=c) return d;
    return a;
}

int max4_3(int a, int b, int c, int d) {
    int m1, m2;
    if(a>b) m1 = a;
    else m1 = b;
    if(c>d) m2 = c;
    else m2 = d;
    if(m1>m2) return m1;
    else return m2;
}
```

Sumo wrestlers in an elevator

- sumo.cpp
- Given the weights of four sumo wrestlers and a an elevator max load, write a function that finds the minimum number of elevator rides to lift all the wrestlers.

```
#include <algorithm>
using namespace std;

// sort 4 numbers so that a >= b >= c >= d
void sort(int& a, int& b, int& c, int& d) {
  if (a < b) swap(a, b);
  if (a < c) swap(a, c);
```

```
  if (a < d) swap(a, d);
  if (b < c) swap(b, c);
  if (b < d) swap(b, d);
  if (c < d) swap(c, d);
}

int sumo1(int a, int b, int c, int d, int L)
{
  // sort weights
  sort(a, b, c, d);

  // number of elevator rides
  int numRides;
  if (a+b+c+d <= L)
    numRides = 1;
  else if ((a+d <= L && b+c <= L) || b+c+d <= L)
    numRides = 2;
  else if (c+d <= L)
    numRides = 3;
  else
    numRides = 4;

  return numRides;
}

int sumo2(int a, int b, int c, int d, int L)
{
  // number of elevator rides
  int numRides;
  if (a+b+c+d <= L)
    numRides = 1;
  else if ((a+b <= L && c+d <= L) ||
   (a+c <= L && b+d <= L) ||
   (a+d <= L && b+c <= L) ||
   (a+b+c <= L /* && d <= L*/ ) ||
   (a+b+d <= L /* && c <= L*/ ) ||
   (a+c+d <= L /* && b <= L*/ ) ||
   (b+c+d <= L /* && a <= L*/ ))
    numRides = 2;
  else if (a+b <= L /* && c <= L && d <= L */ ||
   a+c <= L /* && b <= L && d <= L */ ||
   a+d <= L /* && b <= L && c <= L */ ||
   b+c <= L /* && a <= L && d <= L */ ||
   b+d <= L /* && a <= L && c <= L */ ||
   c+d <= L /* && a <= L && b <= L */)
    numRides = 3;
```

```
  else
    numRides = 4;

  return numRides;
}
```

Aggregate state of water

- water.c
- Write a function that find the aggregate state of water, given its temperature.

```
enum state {ICE, LIQUID, AERATED};

int water1(int celsius) {
    if(celsius <= 0) return ICE;
    if(celsius < 100) return LIQUID;
    if(celsius >= 100) return AERATED;
    return ICE;
}

int water2(int celsius) {
    enum state s = ICE;
    if(celsius > 0 && celsius < 100) s = LIQUID;
    else if(celsius >= 100) s = AERATED;
    return s;
}

int water3(int celsius) {
    enum state s;
    if(celsius <= 0) s = ICE;
    else if(celsius > 0 && celsius < 100) s = LIQUID;
    else if(celsius >= 100) s = AERATED;
    return s;
}

int water4(int celsius) {
    enum state s;
    if(celsius >= 100) s = AERATED;
    else if(celsius > 0 && celsius < 100) s = LIQUID;
    else if(celsius <= 0) s = ICE;
    return s;
}

int water5(int celsius) {
```

```
    enum state s;
    if(celsius <= 0) s = ICE;
    if(celsius > 0 && celsius < 100) s = LIQUID;
    if(celsius >= 100) s = AERATED;
    return s;
}
```

Grade on the exam

- grade.c
- Students are having a test. Those that score less than 51 points get a failing grade 5, those between 51 and 60 points get a grade 6, between 61 and 70 get a grade 7, between 71 and 80 get a grade 8, between 81 and 90 get a grade 9, and those who score above 90 points get the maximal grade 10, Write a function that calculates the grade given the number of points.

```
int grade1(int points) {
  int grade = 0;
  if (points < 51)
    grade = 5;
  else if (points < 61)
    grade = 6;
  else if (points < 71)
    grade = 7;
  else if (points < 81)
    grade = 8;
  else if (points < 91)
    grade = 9;
  else
    grade = 10;

 return grade;
}

int grade2(int points) {
  if (points < 51)
    return 5;
  if (51 <= points && points <= 60)
    return 6;
  if (61 <= points && points <= 70)
    return 7;
  if (71 <= points && points <= 80)
    return 8;
```

```
  if (81 <= points && points <= 90)
    return 9;
  if (91 <= points && points <= 100)
    return 10;
  return 0;
}

int grade3(int points) {
  return (((points - 1) / 10 + 1) < 5 ? 5 : (points - 1) / 10 + 1);
}
```

Uninterpreted functions

Squares of numbers

- square.c
- Write a function that calculates squares of all numbers in a given array.

```
int square1(int a[], int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += a[i]*a[i];
  return s;
}

int square(int a) { return a*a; }

int square2(int a[], int n) {
  int i = 0, s = 0;
  while (i<n) {
    s = s + square(a[i]);
    i = i + 1;
  }
  return s;
}
```

Next round of competition

- competition.c
- Students are qualified for the next round of competition of their score is equal or above the limit. Write a function that given the array of points, counts the number of those who qualified.

```c
int competition1(int a[], int b[], int n, int points) {
  int i, j;
  for (i = 0, j = 0; i < n; i++)
    if(a[i] > points)
        b[j++] = a[i];
  return j;
}

int competition2(int a[], int b[], int n, int points) {
  int i, j = 0;
  for (i = 0; i < n; i++) {
    if(a[i] <= points)
      continue;
    b[j] = a[i];
    j = j + 1;
  }
  return j;
}
```

Date

- date.c
- Write a function that counts the number of dates in an array that are after the given date.

```c
struct date {
    int day;
    int month;
    int year;
};

int greater_dates1(struct date a[], struct date b[], struct date d, int n) {
  int i, j;
  for (i = 0, j = 0; i < n; i++)
    if((a[i].year > d.year) ||
         ((a[i].year == d.year) && (a[i].month > d.month)) ||
         ((a[i].year == d.year) && (a[i].month == d.month)  && (a[i].day > d.day))  ) {
        b[j] = a[i];
        j = j + 1;
    }
  return j;
}
```

```c
int compare_dates (struct date d1, struct date d2)
{
    if (d1.year < d2.year)
       return -1;

    else if (d1.year > d2.year)
       return 1;

    if (d1.year == d2.year)
    {
         if (d1.month<d2.month)
              return -1;
         else if (d1.month>d2.month)
              return 1;
         else if (d1.day<d2.day)
              return -1;
         else if(d1.day>d2.day)
              return 1;
         else
              return 0;
    }
    return 0;
}
int greater_dates2(struct date a[], struct date b[], struct date d, int n) {
  int i, j = 0;
  for (i = 0; i < n; i++)
        if(compare_dates(a[i], d) > 0)
            b[j++] = a[i];
  return j;
}
```

Students

- students.c
- A student is excellent if his GPA is above 4.5. Given an array of GPAs write a function that counts excellent students.

```c
int students1(float a[], float b[], int n) {
  int i, j;
  const float grade_average = 4.5;
  for (i = 0, j = 0; i < n; i++)
    if(a[i] > grade_average) {
        b[j] = a[i];
```

```
        j = j + 1;
    }
  return j;
}

int students2(float a[], float b[], int n) {
  int i, j = 0;
  for (i = 0; i < n; i++)
    (a[i] > 4.5 ? b[j++]=a[i] : 0);
  return j;
}
```

Sum

- sum.c
- Write a function that calculates the sum of all elements in a given array.

```
int sum1(int a[], int n) {
  int i, sum = 0;
  for (i = 0; i < n; i++)
      sum += a[i];
  return sum;
}

int sum2(int a[], int n) {
  int i=0, sum = 0;
  while(i < n) {
    sum = sum + a[i];
    i = i + 1;
  }
  return sum;
}

#ifdef __TESTING__
#include <stdio.h>

#define MAX 100
int main() {
  int n;
  int a[MAX];
  scanf("%d", &n);
  for (int i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("%d\n", sum1(a, n));
```

```
  printf("%d\n", sum2(a, n));
  return 0;
}
#endif
```

Sum of a segment

- sum_ij.c
- Write a function that calculates the sum of elements in a given array stored between the positions i and j.

```
int sum1(int a[], int n, int j, int k) {
  int i, sum = 0;
  for (i = j; i < k; i++)
      sum += a[i];
  return sum;
}

int sum2(int a[], int n, int j, int k) {
  int sum = 0;
  while(j < k)
    sum = sum + a[j++];
  return sum;
}

#ifdef __TESTING__
#include <stdio.h>
#define MAX 100
int main() {
  int a[MAX];
  int n, i, j, k;
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  scanf("%d", &j);
  scanf("%d", &k);
  printf("%d\n", sum1(a, n, j, k));
  printf("%d\n", sum2(a, n, j, k));
  return 0;
}
#endif
```

Factorial

- fact.c
- Write a function that calculates the factorial of a given number.

```
int fact1(int n) {
  int i, p = 1;
  for (i = 1; i <= n; i++)
      p *= i;
  return p;
}

int fact2(int n) {
  int i = 1, p = 1;
  while(i<=n) {
    p *= i;
    i = i + 1;
  }
  return p;
}

int fact3(int n) {
  int i = n, p = 1;
  while(i>0) {
    p *= i;
    i = i - 1;
  }
  return p;
}

int fact4(int n) {
  int i = n, p = 1;
  for (i = n; i > 0; i--)
      p *= i;
  return p;
}

int fact5(int n) {
   int p = 1;
   while(n > 0)
      p *= n--;
   return p;
}

#ifdef __TESTING__
```

```c
#include <stdio.h>

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact1(n));
  printf("%d\n", fact2(n));
  printf("%d\n", fact3(n));
  printf("%d\n", fact4(n));
  printf("%d\n", fact5(n));
  return 0;
}

#endif
```

Geometric mean

- geo.c
- Write a function that calculates the geometric mean of given numbers (for n numbers, the mean is the n-th root of their product).

```c
#include <math.h>

double geo1(int a[], int n) {
  int i, p = 1;
  for (i = 0; i < n; i++)
      p *= a[i];
  return pow(p, 1./n);
}

double geo2(int a[], int n) {
  int i = 0, p = 1;
  while (i < n) {
    p = p * a[i];
    i = i + 1;
  }
  return pow(p, 1./n);
}

#ifdef __TESTING__
#include <stdio.h>
#define MAX 100
int main() {
  int a[MAX];
  int n, i;
```

```
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("%lf\n", geo1(a, n));
  printf("%lf\n", geo2(a, n));
  return 0;
}
#endif
```

Geometric mean of a segment

- geo_ij.c
- Write a function that calculates the geometric mean of numbers stored at position between i and j in a given array.

```
#include <math.h>

double geo1(int a[], int n, int j, int k) {
  int i, p;
  if (k == j) return 1;
  p = 1;
  for (i = j; i < k; i++)
      p *= a[i];
  return pow(p,1./(k-j));
}

double geo2(int a[], int n, int j, int k) {
  int p;
  n = k - j;
  if (n == 0) return 1;
  p = 1;
  while (j < k)
    p = p * a[j++];
  return pow(p,1./n);
}

#ifdef __TESTING__
#define MAX 100
#include <stdio.h>

int main() {
  int n, i, j, k;
  int a[MAX];
  scanf("%d", &n);
  for (i = 0; i < n; i++)
```

```
    scanf("%d", &a[i]);
  scanf("%d", &j);
  scanf("%d", &k);
  printf("%lf\n", geo1(a, n, j, k));
  printf("%lf\n", geo2(a, n, j, k));
  return 0;
}
#endif
```

Sum of digits

- sum_digits.c
- Write a function that calculates the sum of digits in a decimal representation of a given number

```
int sum_digits1(unsigned n) {
  int sum = 0;
  while(n>0) {
      sum += n%10;
      n /= 10;
  }
  return sum;
}

int sum_digits2(unsigned n) {
  int sum = 0, digit;
  while(n>0) {
      digit = n%10;
      sum = sum + digit;
      n = n/10;
  }
  return sum;
}

#ifdef __TESTING__
#include <stdio.h>
int main() {
  unsigned n;
  scanf("%d", &n);
  printf("%d\n", sum_digits1(n));
  printf("%d\n", sum_digits2(n));
  return 0;
}
#endif
```

Number of digits

- count_digits.c
- Write a function that calculates the sum of digits in a decimal representation of a given number.

```
int count1(int n) {
   int count = 0;
   if (n == 0)
     return 1;
   while (n > 0) {
      n /= 10;
      count++;
   }
   return count;
}

int count2(int n) {
   int count, copy = n;
   for (count = 0; copy>0; count++)
      copy /= 10;
   if (n == 0)
     return 1;
   return count;
}

#ifdef __TESTING__
#include <stdio.h>
int main() {
   int n;
   scanf("%d", &n);
   printf("%d\n", count1(n));
   printf("%d\n", count2(n));
   return 0;
}
#endif
```

Check sorted

- is_sorted.c
- Write a function that checks if a given array is sorted.

```
int is_sorted1(int a[], int n) {
  int i;
  for (i = 0; i < n-1; i++)
    if (a[i] >= a[i+1])
```

```
      return 0;
  return 1;
}

int is_sorted2(int a[], int n) {
  int i = 0;
  while (i < n-1)
    if (a[i] < a[i+1])
      i++;
    else
      return 0;
  return 1;
}

#ifndef __TESTING__
#define MAX 100
int main() {
  int n, a[MAX], i;
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  scanf("%d", &k);
  printf("%d\n", is_sorted1(a, n));
  printf("%d\n", is_sorted2(a, n));
  return 0;
}
#endif
```

Check adjacent elements

- check_adjacent.c
- Write a function that checks if each element of a given array is two times less than its predecesor.

```
int check_adjacent1(int a[], int n) {
  int i;
  for (i = 0; i < n - 1; i++)
    if (a[i] > 2 * a[i + 1])
      return 0;
  return 1;
}

int check_adjacent2(int a[], int n) {
  int i = 0;
  while (i < n - 1) {
```

```
    if (a[i + 1] + a[i + 1] < a[i])
      return 0;
    i++;
  }
  return 1;
}

#ifdef __TESTING__
#include <stdio.h>

#define MAX 100

int main() {
  int n, i, a[MAX];
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("%d\n", check_adjacent1(a, n));
  printf("%d\n", check_adjacent2(a, n));
  return 0;
}
#endif
```

Equal

- equal.c
- Write a function that checks if an array contains a given element.

```
int equal1(int a[], int n, int x) {
  int i;
  for (i = 0; i < n; i++)
    if (a[i] != x)
      return 0;
  return 1;
}

int equal2(int a[], int n, int x) {
  int i = 0;
  while (i < n) {
    if (a[i]-x)
      return 0;
    i++;
  }
  return 1;
}
```

```c
#ifdef __TESTING__
#include <stdio.h>

#define MAX 100

int main() {
  int n, i, x, a[MAX];
  scanf("%d", &n);
  for (int i = 0; i < n; i++)
    scantf("%d", &a[i]);
  scanf("%d", &x);
  printf("%d\n", equal1(a, n, x));
  printf("%d\n", equal2(a, n, x));
  return 0;
}
#endif
```

Find first divisible

- find_first_div_k.c
- Write a function that finds the first position of an element in a given array that is divisible by a given number (return -1 if there is no such element).

```c
int find_first_div_k1(int a[], int n, int k) {
  int i;
  for (i = 0; i < n; i++)
    if (a[i] % k == 0)
      return i;
  return -1;
}

int find_first_div_k2(int a[], int n, int k) {
  int i = 0;
  while (i < n)
    if (a[i]%k)
      i++;
    else
      return i;
  return -1;
}

#ifndef __TESTING__
#define MAX 100
int main() {
```

```
   int n, a[MAX], k, i;
   scanf("%d", &n);
   for (i = 0; i < n; i++)
     scanf("%d", &a[i]);
   scanf("%d", &k);
   printf("%d\n", find_first_div_k1(a, n, k));
   printf("%d\n", find_first_div_k2(a, n, k));
   return 0;
}
#endif
```

Search for positive element

- positive.c
- Write a function that checks if there is a positive element in a given array.

```
int positive1(int a[], int n) {
  int i;
  for (i = 0; i < n; i++)
    if (a[i] > 0)
      return 1;
  return 0;
}

int positive2(int a[], int n) {
  int i = 0;
  while (i < n)
    if (a[i] > 0)
      return 1;
    else
      i++;
  return 0;
}

#ifdef __TESTING__
#define MAX 100
int main() {
   int n, a[MAX], i;
   scanf("%d", &n);
   for (i = 0; i < n; i++)
      scanf("%d", &a[i]);
   printf("%d\n", positive1(a, n));
   printf("%d\n", positive2(a, n));
   return 0;
}
```

```
#endif
```

Remainder division by 2 and 3

- r2m3.c
- Write a function that returns the first position of an element in an array that gives the remainder different than 2 when divided by 3 or the remainder different than 1 when divided by 2.

```
int r2m31(int a[], int n) {
  int i;
  for (i = 0; i < n; i++)
    if (a[i] % 3 != 2 || a[i] % 2 != 1)
      return i;
  return -1;
}

int r2m32(int a[], int n) {
  int i;
  for (i = 0; i < n; i++)
    if (a[i] % 6 != 5)
      return i;
  return -1;
}

#ifdef __TESTING__
#include <stdio.h>

#define MAX 100

int main() {
  int n, i, a[MAX];
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("%d\n", r2m31(a, n));
  printf("%d\n", r2m32(a, n));
  return 0;
}
#endif
```

Position of maximum

- max_pos.c
- Write a function that finds the first position of the maximal element of a given array of integers.

```
int maximum1(int a[], int n) {
  int i, pos = 0;
  for (i = 1; i < n; i++)
    if (a[pos] < a[i])
      pos = i;
  return pos;
}

int maximum2(int a[], int n) {
  int i, m = a[0], pos = 0;
  for (i = 1; i < n; i++)
    if (m < a[i]) {
      m = a[i];
      pos = i;
    }
  return pos;
}
```

Minimum

- min.c
- Write a function that finds the value of the minimal element of a given array of integers.

```
int mini(int a, int b) {
  if (a < b)
    return a;
  return b;
}

int minimum1(int a[], int n) {
  int i, m = a[0];
  for (i = 1; i < n; i++)
    if (m > a[i])
      m = a[i];
  return m;
}

int minimum2(int a[], int n) {
  int i, m = a[0];
```

```
  for (i = 1; i < n; i++)
    m = mini(a[i], m);
  return m;
}
```

Minimum of a segment

- min_ij.c
- Write a function that finds the value of the minimum of all elements between positions i and j in a given array of integers.

```
int mini(int a, int b) {
  if (a < b)
    return a;
  return b;
}

int minimum1(int a[], int n, int j, int k) {
  int i, m = a[j];
  for (i = j+1; i < k; i++)
    if (m > a[i])
      m = a[i];
  return m;
}

int minimum2(int a[], int n, int j, int k) {
  int m = a[j++];
  while (j < k)
    m = mini(a[j++], m);
  return m;
}
```

Euclidean distance

- euclid.c
- Write a function that finds the Euclidean norm of a given vector (the square root of the sum of squares of its elements).

```
#include<math.h>

double euclid1(int a[], int n) {
```

```
  int i, sum = 0;
  for (i = 0; i < n; i++)
      sum += a[i]*a[i];
  return sqrt(sum);
}

int my__square(int a) {
    return a*a;
}

double euclid2(int a[], int n) {
  int i = 0, sum = 0;
  while (i < n) {
    sum = sum + my__square(a[i]);
    i = i + 1;
  }
  return sqrt(sum);
}
```

Filter sum

- filter_sum.c
- Write a function that calculates the sum of all elements in a given array that are different from the given one.

```
int filter_sum1(int a[], int n, int x) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    if (a[i] != -x)
      s += -x;
  return s;
}

int filter_sum2(int a[], int n, int x) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    (a[i] + x ? s += -x : 0);
  return s;
}
```

Filter product

- filter_product.c
- Write a function that calculates the product of all elements in a given array that are different from the given one.

```
int product_not_equal1(int a[], int n, int x) {
  int i, p = 1;
  for (i = 0; i < n; i++)
    if (a[i]-x) p*=a[i];
  return p;
}

int product_not_equal2(int a[], int n, int x) {
  int i=0, p = 1;
  while(i<n) {
    if(a[i] != x)
      p*=a[i];
    i += 1;
  }
  return p;
}
```

Max even

- max_even.c
- Write a function that finds the maximal even number in an array.

```
int max_even1(int a[], int n) {
  int i, max = -1, found = 0;
  for (i = 0; i < n; i++) {
    if (a[i] % 2 == 0)
      if (!found || a[i] > max) {
        found = 1;
        max = a[i];
      }
  }
  return max;
}


#define max2(a, b) ((a) >= (b) ? (a) : (b))

int max_even2(int a[], int n) {
```

```
  int i = 0;
  int max = -1;
  int found = 0;
  while(i<n) {
    if (!(a[i] % 2))
      if (found == 0) {
           max = a[i];
           found = 1;
       } else
        max = max2(max, a[i]);
    i += 1;
  }
  return max;
}
```

Number positive

- number_positive.c
- Write a function that counts positive elements in a given array.

```
int number_positive1(int a[], int n) {
  int i, num = 0;
  for (i = 0; i < n; i++)
    if (a[i] > 0)
        num++;
  return num;
}

int number_positive2(int a[], int n) {
  int i=0, num = 0;
  while(i<n)
    num += (a[i++] > 0 ? 1 : 0) ;
  return num;
}
```

Bottle packing

- pack.c
- There are n shelves in a store, and each shelf contains a different kind of bottles with juice. From each shelf bottles are packed in boxes that can store k bottles. If a number of bottles on each shelf is know, and boxes

can contain only a single kind of juice, write a function that calculates the minimal number of boxes to store all the bottles.

```
int pack1(int a[], int n, int k) {
  int s = 0, i;
  for (i = 0; i < n; i++) {
    s += a[i] / k;
    if (a[i] % k != 0)
      s++;
  }
  return s;
}

int pack2(int a[], int n, int k) {
  int s = 0, i;
  for (i = 0; i < n; i++) {
    int b = a[i] % k == 0 ? a[i] / k : a[i] / k + 1;
    s = s + b;
  }
  return s;
}
```

Product of squares of even numbers

- product_square_even.c
- Write a function that calculates the product of squares of all even numbers in an array.

```
int product_square_even1(int a[], int n) {
  int i, p = 1;
  for (i = 0; i < n; i++)
    if ((a[i]%2) == 0) p*=(a[i]*a[i]);
  return p;
}

#define sqr(a) ((a)*(a))
int product_square_even2(int a[], int n) {
  int i=0, p = 1;
  while(i<n) {
    if(!(a[i]%2))
      p*=sqr(a[i]);
    i += 1;
  }
  return p;
```

```
}
```

Sum of absolute values

- sum_abs.c
- Write a function that calculates the sum of absolute values of all elements in an array.

```c
#include <stdlib.h>

int sum_abs1(int a[], int n) {
  int i, sum = 0;
  for (i = 0; i < n; i++)
    if (a[i] > 0)
      sum += a[i];
    else
      sum += -a[i];
  return sum;
}

int sum_abs2(int a[], int n) {
  int i, sum = 0;
  for (i = 0; i < n; i++)
      sum += abs(a[i]);
  return sum;
}

int myabs(int a) {
  if (a >= 0)
    return a;
  return -a;
}

int sum_abs3(int a[], int n) {
  int i = 0, sum = 0;
  while (i < n)
    sum += myabs(a[i++]);
  return sum;
}
```

Sum of elements less than a given value

- sum_less_than.c
- Write a function that calculates the sum of all elements in an array that are less than the given one.

```
int sum_less_than1(int a[], int n, int x) {
  int i, sum = 0;
  for (i = 0; i < n; i++)
    if (a[i]<x) sum+=a[i];
  return sum;
}

int sum_less_than2(int a[], int n, int x) {
  int i=0, sum = 0;
  while(i<n) {
    sum += (a[i] < x ? a[i] : 0);
    i += 1;
  }
  return sum;
}
```

Sum of squares of odd digits

- sum_square_odd_digits.c
- Write a function that calculates the sum of squares of all odd digits in a decimal representation of a given natural number.

```
#include<math.h>

int sum1(unsigned n) {
  int sum = 0, d;
  while(n>0) {
    d = n%10;
    if(d%2) sum+=d*d;
    n=n/10;
  }
  return sum;
}

int sum2(unsigned n) {
  int sum;
  for(sum = 0; n > 0; n /= 10) {
    sum += ((n%10)%2 ? (n%10)*(n%10) : 0);
```

```
  }
  return sum;
}
```

K-equivalence

Search

- binarySearch.cpp
- Given a sorted, distinct array find the position of a given element x, or return -1 if it is not present in the list.

```
int linearSearch(int a[], int n, int x) {
  for (int i = 0; i < n; i++)
    if (a[i] == x)
      return i;
  return -1;
}

int binarySearchRec_(int a[], int l, int r, int x) {
  if (l > r) return -1;
  int s = l + (r - l) / 2;
  if (x < a[s])
    return binarySearchRec_(a, l, s-1, x);
  if (x > a[s])
    return binarySearchRec_(a, s+1, r, x);
  return s;
}

int binarySearchRec(int a[], int n, int x) {
  return binarySearchRec_(a, 0, n-1, x);
}

int binarySearchEq(int a[], int n, int x) {
  int l = 0, r = n-1;
  while (l <= r) {
    int s = l + (r - l) / 2;
    if (x < a[s])
      r = s-1;
    else if (x > a[s])
      l = s+1;
    else
      return s;
  }
```

```
  return -1;
}

int binarySearchGeq1(int a[], int n, int x) {
  int l = 0, r = n-1;
  while (l <= r) {
    int s = l + (r - l) / 2;
    if (a[s] < x)
      l = s+1;
    else
      r = s-1;
  }
  if (l < n && a[l] == x)
    return l;
  return -1;
}

int binarySearchGeq2(int a[], int n, int x) {
  int l = 0, r = n;
  while (l < r) {
    int s = l + (r - l) / 2;
    if (a[s] < x)
      l = s+1;
    else
      r = s;
  }
  return r < n && a[r] == x ? r : -1;
}

#ifdef __TESTING__

#include <iostream>
using namespace std;

int main() {
  int a[] = {1, 3, 4, 5, 7, 9, 11, 13, 18};
  int n = sizeof(a) / sizeof(int);
  int x = 13;
  cout << linearSearch(a, n, x) << endl;
  cout << binarySearchRec(a, n, x) << endl;
  cout << binarySearchEq(a, n, x) << endl;
  cout << binarySearchGeq1(a, n, x) << endl;
  cout << binarySearchGeq2(a, n, x) << endl;
  return 0;
}
```

```
#endif
```

Sort

- sorting.cpp
- Write a program that sorts a given array.

```
void swap(int a[], int i, int j) {
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}

void selectionSortPosMinimum(int a[], int n) {
  for (int i = 0; i < n - 1; i++) {
    int pos = i;
    for (int j = i+1; j < n; j++) {
      if (a[j] < a[pos])
    pos = j;
    }
    if (pos != i)
      swap(a, pos, i);
  }
}

void selectionSortSwaps(int a[], int n) {
  for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
      if (a[i] > a[j])
    swap(a, i, j);
}

void insertionSortSwaps(int a[], int n) {
  for (int i = 1; i < n; i++) {
    int tmp = a[i];
    int j;
    for (j = i - 1; j >= 0 && a[j] > tmp; j--)
      swap(a, j, j+1);
    a[j+1] = tmp;
  }
}

void insertionSortOptimized(int a[], int n) {
   for (int i = 1; i < n; i++) {
      int tmp = a[i];
```

```
      int j;
      for (j = i - 1; j >= 0 && a[j] > tmp; j--) {
        int tmp = a[j]; a[j] = a[j+1]; a[j+1] = tmp;
      }
      a[j+1] = tmp;
   }
}

void quickSort11(int a[], int l, int d) {
  if (l < d) {
    int pivot = a[l];
    int last_leq_pivot = l;
    for (int i = l + 1; i <= d; i++)
      if (a[i] <= pivot)
    swap(a, i, ++last_leq_pivot);
    swap(a, l, last_leq_pivot);
    quickSort11(a, l, last_leq_pivot-1);
    quickSort11(a, last_leq_pivot+1, d);
  }
}

void quickSort22(int a[], int l, int d) {
  if (l < d) {
    int pivot = a[l];
    int i = l+1, j = d;
    while (i <= j) {
      if (a[i] <= pivot)
    i++;
      else if (a[j] > pivot)
    j--;
      else {
    swap(a, i, j);
    i++; j--;
      }
    }
    swap(a, l, j);
    quickSort22(a, l, j-1);
    quickSort22(a, j+1, d);
  }
}

void quickSort1(int a[], int n) {
  quickSort11(a, 0, n-1);
}

void quickSort2(int a[], int n) {
```

```
  quickSort22(a, 0, n-1);
}

void mergeSort1(int a[], int l, int d, int b[]) {
  if (l < d) {
    int s = l + (d - l) / 2;
    mergeSort1(a, l, s, b);
    mergeSort1(a, s+1, d, b);
    int i = l, j = s + 1, k = l;
    while (i <= s && j <= d) {
      if (a[i] <= a[j])
    b[k++] = a[i++];
      else
    b[k++] = a[j++];
    }
    while (i <= s)
      b[k++] = a[i++];
    while (j <= d)
      b[k++] = a[j++];
    for (int i = l; i <= d; i++)
      a[i] = b[i];
  }
}

void mergeSort(int a[], int n) {
  int b[MAX];
  mergeSort1(a, 0, n-1, b);
}


#ifdef __TESTING__

#include <iostream>
using namespace std;


void print(int a[], int n) {
  for (int i = 0; i < n; i++)
    cout << a[i] << " ";
  cout << endl;
}

int main() {
   int a1[] = {5, 3, 8, 2, 4, 6, 1, 7, 9, 0};
   int n1 = sizeof(a1) / sizeof(int);
   mergeSort(a1, n1);
```

```
   print(a1, n1);
   return 0;
}

#endif
```

k-th element

- kthElement.cpp
- Write a program that determines k-th element by size (k-th element in a non-decreasing order) in a given array of integers.

```
void swap(int a[], int i, int j) {
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}

int partition(int a[], int l, int r) {
  int pivot = a[l];
  int last_leq_pivot = l;
  for (int i = l+1; i <= r; i++)
    if (a[i] <= pivot)
      swap(a, i, ++last_leq_pivot);
  swap(a, l, last_leq_pivot);
  return last_leq_pivot;
}

int kthElementQuickSelect(int a[], int n, int k) {
  int l = 0, r = n-1;
  while (l <= r) {
    int pivot_pos = partition(a, l, r);
    if (k < pivot_pos)
      r = pivot_pos-1;
    else if (k > pivot_pos)
      l = pivot_pos+1;
    else
      return a[pivot_pos];
  }
  return -1;
}

void quickSort_(int a[], int l, int r) {
  if (l >= r) return;
  int pivot_pos = partition(a, l, r);
```

```
  quickSort_(a, l, pivot_pos-1);
  quickSort_(a, pivot_pos+1, r);
}

void quickSort(int a[], int n) {
  return quickSort_(a, 0, n-1);
}

int kthElementSelectionSort(int a[], int n, int k) {
  for (int i = 0; i < n; i++)
    for (int j = i+1; j<n; j++)
      if (a[i] > a[j])
    swap(a, i, j);
  return a[k];
}

int kthElementQuickSort(int a[], int n, int k) {
  quickSort(a, n);
  return a[k];
}

#ifdef __TESTING__

#include <iostream>
using namespace std;

int main() {
  int a[] = {3, 5, 8, 4, 2, 6, 7, 0, 1, 16, 13, 10};
  int n = sizeof(a) / sizeof(int);
  for (int k = 0; k < n; k++) {
    cout << kthElementSelectionSort(a, n, k) << " ";
    cout << kthElementQuickSort(a, n, k) << " ";
    cout << kthElementQuickSelect(a, n, k) << " ";
    cout << endl;
  }
}

#endif
```

Majority element

- majority.cpp
- A majority element is the one that occurs more than all other elements together. Write a program that finds a majority element in the given array of positive integers (if there is no such element report -1).

```
int majorityCounting(int a[], int n) {
   int count[MAX] = {0};
   for (int i = 0; i < n; i++)
     count[a[i]]++;
   for (int x = 0; x < MAX; x++)
     if (count[x] > n/2)
        return x;
   return -1;
}

void swap(int a[], int i, int j) {
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}

int partition(int a[], int l, int r) {
  int pivot = a[l];
  int last_leq_pivot = l;
  for (int i = l+1; i <= r; i++)
    if (a[i] <= pivot)
      swap(a, i, ++last_leq_pivot);
  swap(a, l, last_leq_pivot);
  return last_leq_pivot;
}

int kthElementQuickSelect(int a[], int n, int k) {
  int l = 0, r = n-1;
  while (l <= r) {
    int pivot_pos = partition(a, l, r);
    if (k < pivot_pos)
      r = pivot_pos-1;
    else if (k > pivot_pos)
      l = pivot_pos+1;
    else
      return a[pivot_pos];
  }
  return -1;
}

int majorityMedian(int a[], int n) {
  int candidate = kthElementQuickSelect(a, n, n/2);
  int count = 0;
  for (int i = 0; i < n; i++)
     if (a[i] == candidate)
```

```
      count++;
  return count > n/2 ? candidate : -1;
}

int majorityBoyerMoore(int a[], int n) {
   int candidate, freq = 0;
   for (int i = 0; i < n; i++)
     if (freq == 0) {
        candidate = a[i]; freq = 1;
     } else {
        if (a[i] == candidate)
          freq++;
        else
          freq--;
     }
   if (freq > 0) {
      int cnt = 0;
      for (int i = 0; i < n; i++)
        if (a[i] == candidate)
          cnt++;
      if (cnt > n/2) return candidate;
   }
   return -1;
}

#ifdef __TESTING__

#include <iostream>
using namespace std;

int main() {
  int a[] = {3, 1, 2, 3, 2, 2, 3, 1, 2, 3, 2, 2, 2, 3, 2, 3, 2};
  int n = sizeof(a) / sizeof(int);
  cout << majorityCounting(a, n) << endl;
  cout << majorityMedian(a, n) << endl;
  cout << majorityBoyerMoore(a, n) << endl;
}

#endif
```

Fibonacci numbers

- Write a program that calculates n-th Fibonnaci number, i.e., n-th element of a sequence $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$ that starts with 0 and 1 and each next number is the sum of the two previous ones.

- fib.cpp

```cpp
int fibRecursive(int n) {
  if (n < 2)
    return n;
  return fibRecursive(n-1) + fibRecursive(n-2);
}

int fibMemo_(int n, int memo[]) {
  if (memo[n] != 0)
    return memo[n];

  if (n < 2)
    return memo[n] = n;

  else return memo[n] = fibMemo_(n-1, memo) + fibMemo_(n-2, memo);
}

int fibMemo(int n) {
  int memo[MAX];
  int i;
  for (i = 0; i <= n; i++)
    memo[i] = 0;
  return fibMemo_(n, memo);
}

int fibDP(int n) {
  int dp[MAX];
  dp[0] = 0;
  dp[1] = 1;
  for (int i = 2; i <= n; i++)
    dp[i] = dp[i-1] + dp[i-2];
  return dp[n];
}

int fibDPMemOptimized(int n) {
  if (n < 2)
    return n;

  int fpp = 0, fp = 1;
  for (int i = 2; i <= n; i++) {
    int f = fpp + fp;
    fpp = fp;
    fp = f;
  }
```

```
  return fp;
}

#ifdef __TESTING__

#include <iostream>
using namespace std;

int main() {
  for (int n = 0; n < 10; n++) {
    cout << fibRecursive(n) << " "
     << fibMemo(n) << " "
     << fibDP(n) << " "
     << fibDPMemOptimized(n) << endl;
  }
  return 0;
}

#endif
```

Longest increasing subsequence

- Write a program that calculates the length of the longest increasing subsequence of a given array (subsequence elements need not be consecutive).
- longestIncreasingSubsequence.cpp

```
void longestIncreasingSubsequenceRecursive_(int a[], int k,
                        int* max_ending_k, int* max) {
  *max_ending_k = 1;

  for (int i = 0; i < k; i++) {
    int max_ending_i;
    longestIncreasingSubsequenceRecursive_(a, i, &max_ending_i, max);
    if (a[i] < a[k] && *max_ending_k < max_ending_i + 1)
      *max_ending_k = max_ending_i + 1;
  }

  if (*max < *max_ending_k)
    *max = *max_ending_k;
}

int longestIncreasingSubsequenceRecursive(int a[], int n) {
  int max_ending, max = 0;
  if (n > 0)
```

```
    longestIncreasingSubsequenceRecursive_(a, n-1, &max_ending, &max);
  return max;
}



void longestIncreasingSubsequenceMemo_(int a[], int k,
                       int* max_ending_k, int* max, int memo[]) {
  if (memo[k] > 0) {
    *max_ending_k = memo[k];
    return;
  }

  *max_ending_k = 1;

  for (int i = 0; i < k; i++) {
    int max_ending_i;
    longestIncreasingSubsequenceMemo_(a, i, &max_ending_i, max, memo);
    if (a[i] < a[k] && *max_ending_k < max_ending_i + 1)
      *max_ending_k = max_ending_i + 1;
  }

  memo[k] = *max_ending_k;
  if (*max < *max_ending_k)
    *max = *max_ending_k;
}

int longestIncreasingSubsequenceMemo(int a[], int n) {
  int memo[MAX];
  for (int i = 0; i < n; i++)
    memo[i] = 0;
  int max_ending, max = 0;
  if (n > 0)
    longestIncreasingSubsequenceMemo_(a, n-1, &max_ending, &max, memo);
  return max;
}

int longestIncreasingSubsequenceDP(int a[], int n) {
  // dp[i] - length of the longest increasing subsequence finishing at a[i]
  int dp[MAX];
  int max = 0;
  for (int i = 0; i < n; i++) {
    dp[i] = 1;
    for (int j = 0; j < i; j++)
      if (a[i] > a[j] && dp[j] + 1 > dp[i])
    dp[i] = dp[j] + 1;
```

```
    if (dp[i] > max)
      max = dp[i];
  }
 return max;
}


// position of the first element greater or equal to x (binary search
// of a sorted array)
int lowerBound(int a[], int l, int d, int x) {
  while (l < d) {
    int s = l + (d - l) / 2;
    if (a[s] < x)
      l = s + 1;
    else
      d = s;
  }
  return l;
}

int longestIncreasingSubsequenceDPOptimized(int a[], int n) {
  // dp[k] - smallest possible ending element of an increasing
  // subsequence of lenght k
  int dp[MAX];
  int max = 0;
  for (int i = 0; i < n; i++) {
    int k = lowerBound(dp, 0, max, a[i]);
    dp[k] = a[i];
    if (k + 1 > max)
      max = k + 1;
  }
  return max;
}

#ifdef __TESTING__

#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

int main() {
  /*
  srand(time(NULL));
  int a[MAX];
  int n = rand() % 10 + 1;
```

```
  for (int i = 0; i < n; i++)
    a[i] = rand() % 100;
  */

  int a[] = {0, 1}, n = 2;
  cout << longestIncreasingSubsequenceRecursive(a, n) << endl;
  cout << longestIncreasingSubsequenceMemo(a, n) << endl;
  cout << longestIncreasingSubsequenceDP(a, n) << endl;
  cout << longestIncreasingSubsequenceDPOptimized(a, n) << endl;
  return 0;
}

#endif
```

Minimal number of coins

- Given values of different coins, and assuming that there is an infinite supply of each kind of coins, find the minimal number of coins to make a given change.
- coinChange.cpp

```
const int maxS = 10;
const int INF = 4;

int min(int a, int b) {
  return a < b ? a : b;
}

int minCoinsRecursive(int v[], int n, int s){
  if (s == 0)
    return 0;
  int num = INF;
  for (int i = 0; i < n; i++)
    if (v[i] <= s)
      num = min(num, minCoinsRecursive(v, n, s-v[i]) + 1);
  return num;
}

int minCoinsMemo_(int v[], int n, int S, int memo[]){
  if (memo[S] != 0)
    return memo[S];
  if (S == 0)
    return memo[S] = 0;
  int br = INF;
  for (int i = 0; i < n; i++)
```

```
    if (v[i] <= S)
      br = min(br, minCoinsMemo_(v, n, S-v[i], memo) + 1);
  return memo[S] = br;
}

int minCoinsMemo(int v[], int n, int S) {
  int memo[maxS + 1];
  for (int i = 0; i <= maxS; i++)
    memo[i] = 0;
  return minCoinsMemo_(v, n, S, memo);
}

int minCoinsDP(int v[], int N, int S){
  int dp[maxS + 1];
  dp[0] = 0;
  for (int s = 1; s <= S; s++) {
    dp[s] = INF;
    for (int n = 0; n < N; n++)
      if (v[n] <= s)
    dp[s] = min(dp[s], dp[s-v[n]] + 1);
  }
  return dp[S];
}

int minCoinsRecursive1(int v[], int n, int s){
  if (s == 0)
    return 0;
  if (n == 0)
    return INF;
  int num = minCoinsRecursive1(v, n-1, s);
  if (s >= v[n-1])
    num = min(num, minCoinsRecursive1(v, n, s - v[n-1]) + 1);
  return num;
}

int minCoinsMemo1_(int v[], int n, int s, int memo[MAX][maxS + 1]){
  if (memo[n][s] != 0)
    return memo[n][s];
  if (s == 0)
    return memo[n][s] = 0;
  if (n == 0)
    return memo[n][s] = INF;
  int num = minCoinsMemo1_(v, n-1, s, memo);
  if (s >= v[n-1])
    num = min(num, minCoinsMemo1_(v, n, s - v[n-1], memo) + 1);
  return memo[n][s] = num;
```

```
}

int minCoinsMemo1(int v[], int n, int s) {
  int memo[MAX][maxS + 1];
  for (int i = 0; i < MAX; i++)
    for (int j = 0; j <= maxS; j++)
      memo[i][j] = 0;
  return minCoinsMemo1_(v, n, s, memo);
}

int minCoinsDP1(int v[], int N, int S){
  int dp[maxS + 1];
  dp[0] = 0;
  for (int s = 1; s <= S; s++)
    dp[s] = INF;

  for (int n = 1; n <= N; n++) {
    for (int s = 0; s <= S; s++) {
      if (s >= v[n-1])
    dp[s] = min(dp[s], dp[s - v[n-1]] + 1);
    }
  }

  return dp[S];
}


#ifdef __TESTING__

#include <iostream>
using namespace std;

int main() {
  int N, S;
  int v[MAX];
  cin >> S >> N;
  for (int i = 0; i < N; i++)
    cin >> v[i];
  int num = minCoinsRecursive1(v, N, S);
  cout << (num == INF ? -1 : num)  << endl;
  return 0;
}

#endif
```

Stock span

- closestLargerPredecesor.cpp
- For each element in a given array of positive integers find its closest predecsor that is greater than it (if there is no such predecesor, return -1). For example, given the array $3, 5, 2, 4, 7, 6, 5, 1$, the program should report $-1, -1, 5, 5, -1, 7, 6, 5$.

```
int closestLargerPredecesorBruteForce(int a[], int n, int b[]) {
  int k = 0;
  for (int i = 0; i < n; i++) {
    int found = 0;
    for (int j = i-1; j >= 0; j--)
      if (a[j] > a[i]) {
    b[k++] = a[j];
    found = 1;
    break;
      }
    if (!found)
      b[k++] = -1;
  }
  return k;
}

int closestLargerPredecesorMaximumsStack(int a[], int n, int b[]) {
  int s[MAX];
  int sp = 0, k = 0;
  for (int i = 0; i < n; i++) {
    while (sp > 0 && s[sp - 1] <= a[i])
      sp--;
    b[k++] = (sp == 0 ? -1 : s[sp - 1]);
    s[sp++] = a[i];
  }
  return k;
}

#ifdef __TESTING__

#include <iostream>
using namespace std;

int main() {
  int a[] = {3, 5, 2, 4, 7, 6, 5, 1};
  int n = sizeof(a) / sizeof(int);
  int b[MAX], k;
  k = closestLargerPredecesorMaximumsStack(a, n, b);
```

```
  for (int i = 0; i < k; i++)
    cout << b[i] << " ";
  cout << endl;
  k = closestLargerPredecesorBruteForce(a, n, b);
  for (int i = 0; i < k; i++)
    cout << b[i] << " ";
  cout << endl;
  return 0;
}

#endif

int numPairsBruteForce(int a[], int n, int d) {
  int num = 0;
  for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++) {
      if (a[j] - a[i] == d)
    num++;
      else if (a[j] - a[i] > d)
    break;
    }
  return num;
}

int BinarySearch(int a[], int l, int r, int x) {
  while (l <= r) {
    int s = l + (r - l) / 2;
    if (a[s] < x)
      l = s + 1;
    else
      r = s - 1;
  }
  return l;
}

int numPairsBinarySearch(int a[], int n, int d) {
  int num = 0;
  int i = 0;
  while(i < n) {
    int ii;
    for (ii = i+1; ii < n && a[ii] == a[i]; ii++)
      ;
    int num_ai = ii - i;
    int j = BinarySearch(a, ii, n-1, a[i] + d);
    if (j < n && a[j] == a[i] + d) {
      int jj;
```

```
      for (jj = j+1; jj < n && a[jj] == a[j]; jj++)
    ;
      int num_aj = jj - j;
      num += num_ai * num_aj;
    }
    i = ii;
  }
  return num;
}



int numPairsTwoPointer(int a[], int n, int d) {
  int num = 0;
  int i = 0, j = 1;
  while (j < n) {
    if (a[j] - a[i] < d)
      j++;
    else if (a[j] - a[i] > d)
      i++;
    else {
      int ii;
      for (ii = i+1; ii < n && a[ii] == a[i]; ii++)
    ;
      int num_ai = ii - i;
      i = ii;
      int jj;
      for (jj = j+1; jj < n && a[jj] == a[j]; jj++)
    ;
      int num_aj = jj - j;
      j = jj;
      num += num_ai * num_aj;
    }
  }
  return num;
}

#ifdef __TESTING__

#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
  srand(time(NULL));
```

```
  const int N = 50;
  int a[N];
  a[0] = rand() % 2;
  for (int i = 1; i < N; i++)
     a[i] = a[i-1] + rand() % 2;
  int d = rand() % N / 2 + 1;

  for (int i = 0; i < N; i++)
    cout << a[i] << " ";
  cout << endl;

  cout << d << endl;

  cout << numPairsBinarySearch(a, N, d) << endl;
  cout << numPairsBruteForce(a, N, d) << endl;
  cout << numPairsTwoPointer(a, N, d) << endl;
  return 0;
}

#endif
```

Maximal segment sum

- maxSubarraySum.cpp
- Write a program that calculates the maximal sum of a segment (subarray of consecutive elements) of a given integer sequence. For example, given an array $-2, -3, 4, -1, -2, 1, 5, -3$, the maximum sum is $4 + (-1) + (-2) + 1 + 5 = 7$.

```
int maxSubArraySumBruteForce(int a[], int n) {
  int max = 0;
  for (int i = 0; i < n; i++) {
    int z = 0;
    for (int j = i; j < n; j++) {
      z += a[j];
      if (z > max)
    max = z;
    }
  }
  return max;
}

int maxSubArraySumTwoPointer(int a[], int n) {
  int max = 0;
  int i = 0;
```

```
  while (i < n) {
    int z = 0;
    int j;
    for (j = i; j < n; j++) {
      z += a[j];
      if (z < 0)
    break;
      if (z > max)
    max = z;
    }
    i = j + 1;
  }
  return max;
}

int maxSubArraySumKadane(int a[], int n) {
  int max_ending_here = 0, max_so_far = max_ending_here;
  for (int i = 0; i < n; i++) {
    max_ending_here += a[i];
    if (max_ending_here < 0)
      max_ending_here = 0;
    if (max_so_far < max_ending_here)
      max_so_far = max_ending_here;
  }
  return max_so_far;
}

int maxSubArraySumPartialSums(int a[], int n) {
  int partial_sum = 0;
  int min_partial_sum = partial_sum;
  int max = 0;
  for (int i = 0; i < n; i++) {
    partial_sum += a[i];
    int segment_sum = partial_sum - min_partial_sum;
    if (segment_sum > max)
      max = segment_sum;
    if (partial_sum < min_partial_sum)
      min_partial_sum = partial_sum;
  }
  return max;
}

int max3(int a, int b, int c) {
  int max = a;
  if (b > max) max = b;
  if (c > max) max = c;
```

```
  return max;
}

int maxSubArraySumDivideAndConquer_(int a[], int l, int d) {
  if (l > d)
    return 0;
  int s = l + (d - l) / 2;
  int max_sum_left = maxSubArraySumDivideAndConquer_(a, l, s-1);
  int max_sum_right = maxSubArraySumDivideAndConquer_(a, s+1, d);
  int sum_middle = a[s];
  int max_sum_middle = sum_middle;
  for (int i = s-1; i >= l; i--) {
    sum_middle += a[i];
    if (sum_middle > max_sum_middle)
      max_sum_middle = sum_middle;
  }
  sum_middle = max_sum_middle;
  for (int i = s+1; i <= d; i++) {
    sum_middle += a[i];
    if (sum_middle > max_sum_middle)
      max_sum_middle = sum_middle;
  }
  return max3(max_sum_left, max_sum_right, max_sum_middle);
}

int maxSubArraySumDivideAndConquer(int a[], int n) {
  return maxSubArraySumDivideAndConquer_(a, 0, n - 1);
}

#ifdef __TESTING__

#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
  int n = 3;
  int a[10];
  srand(time(NULL));
  for (int i = 0; i < n; i++)
    a[i] = rand() % 200 - 100;
  cout << maxSubArraySumBruteForce(a, n) << endl;
  cout << maxSubArraySumTwoPointer(a, n) << endl;
  cout << maxSubArraySumKadane(a, n) << endl;
  cout << maxSubArraySumPartialSums(a, n) << endl;
```

```
  cout << maxSubArraySumDivideAndConquer(a, n) << endl;
  return 0;
}

#endif
```

Number of segments of natural numbers with a given sum

- numOfSegmentsWithGivenSum.cpp
- Given an array of positive naturals, determine the number of segments (subarrays of consecutive elements) with sum equal to the given number s.

```
int numOfSegmentsWithGivenSumBruteForce(int a[], int n, int x) {
  int b = 0;
  for (int i = 0; i < n; i++) {
     int z = 0;
     for (int j = i; j < n; j++) {
        z += a[j];
    if (z >= x) {
      if (z == x)
        b++;
      break;
    }
     }
  }
  return b;
}

int numOfSegmentsWithGivenSumTwoPointer(int a[], int n, int x) {
   int b = 0;
   int i = 0, j = 0, z = a[0];
   while (1) {
      if (z < x) {
         j++;
         if (j == n) break;
         z += a[j];
      } else {
         if (z == x) b++;
         z -= a[i]; i++;
      }
   }
   return b;
}
```

```
int numOfSegmentsWithGivenSumPartialSumsTwoPointer(int a[], int n, int x) {
  int b[MAX];
  b[0] = 0;
  for (int i = 1; i <= n; i++)
    b[i] = b[i-1] + a[i-1];

  int num = 0;
  int l = 0, r = 1;
  while (r <= n) {
    if (b[r] - b[l] < x)
      r++;
    else if (b[r] - b[l] > x)
      l++;
    else {
      num++;
      l++;
      r++;
    }
  }
  return num;
}

#ifdef __TESTING__

#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

int main() {
  int a[MAX];
  int n = rand() % (MAX - 1) + 1;
  srand(time(NULL));
  for (int i = 0; i < MAX; i++) {
    a[i] = rand() % 10 + 1;
    cout << a[i] << " ";
  }
  cout << endl;
  int x = 10;
  cout << numOfSegmentsWithGivenSumPartialSumsTwoPointer(a, n, x) << endl;;
  cout << numOfSegmentsWithGivenSumBruteForce(a, n, x) << endl;;
  cout << numOfSegmentsWithGivenSumTwoPointer(a, n, x) << endl;;
  return 0;
}

#endif
```

Number of pairs of elements with a given sum in a sorted, distinct array

- numPairsWithGivenSum.cpp
- Given a sorted array containing distinct elements and a number s write a program that calculates how many pairs of array elements sum up to s.

```
int numPairsBruteForce(int a[], int n, int s) {
  int numPairs = 0;
  for (int i = 0; i < n - 1; i++)
    for(int j = i + 1; j < n; j++)
      if (a[i] + a[j] == s)
        numPairs++;
  return numPairs;
}

int numPairsTwoPointer(int a[], int n, int s) {
  int numPairs = 0;
  int left = 0, right = n - 1;
  while (left < right)
    if (a[left] + a[right] > s)
      right--;
    else if (a[left] + a[right] < s)
      left++;
    else {
      numPairs++;
      left++;
      right--;
  }
  return numPairs;
}


int binarySearch(int a[], int l, int r, int x) {
  while (l <= r) {
    int s = l + (r - l) / 2;
    if (x < a[s])
      r = s-1;
    else if (x > a[s])
      l = s+1;
    else
      return 1;
  }
  return 0;
}
```

```
int numPairsBinarySearch(int a[], int n, int s) {
  int brojParova = 0;
  for (int i = 0; i < n - 1; i++)
    if (binarySearch(a, i+1, n-1, s - a[i]))
      brojParova++;
  return brojParova;
}

#ifdef __TESTING__

#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

int main() {
  const int N = 50;
  srand(time(NULL));
  int a[N];
  a[0] = rand() % 2 + 1;
  for (int i = 1; i < N; i++)
    a[i] = a[i-1] + rand() % 2 + 1;
  int s = rand() % a[N-1] + 1;
  cout << numPairsBruteForce(a, N, s) << endl;
  cout << numPairsBinarySearch(a, N, s) << endl;
  cout << numPairsTwoPointer(a, N, s) << endl;
  return 0;
}

#endif
```

Number of pairs of elements with a given difference in a sorted array

- numPairsWithGivenDifference.cpp
- Given a sorted array and a nonnegative number d, write a program that counts pairs of array elements that have difference d.

Optimal pairing

- A set of n pupils is learning programming. Each pupil can become a menthor to at most one apprentice and can be an apprentice to at most one menthor. A pupil can be a menthor an apprentice if the menthors

rating is at least double than apprentices rating. Given a sorted array of ratings, write a program that determines the highest numer of menthor-apprentice that can be formed. For example, if the ratings are $1020, 1840, 2760, 3990, 4540, 7340$, three pairs can be formed (e.g., 1020 can be an apprentice to 2760, that can be apprentice to 7340, while 1840 can be an apprentice to 4540).

- optimalPairing.cpp

```
int maxPairingGreedyWeakestPupilsWeakestMenthors(int rating[], int n)  {
  // number of pupil-menthor pairs
  int numPairs = 0;
  // indicates if the person already became a menthor
  int isMenthor[MAX];
  for (int i = 0; i < n; i++)
    isMenthor[i] = 0;

  // we assign menthors to pupils, starting from the weakest pupils
  // each pupil is assigned a weakest possible unassigned menthor
  for (int pupil = 0, menthor = 1; menthor < n; pupil++)
    // while there is a potential menthor for the current pupil
    while (menthor < n)
      if (!isMenthor[menthor] && rating[menthor] >= 2 * rating[pupil]) {
        // we found a menthor for the current pupil
        numPairs++;
        isMenthor[menthor] = 1;
        break;
      } else
        // we consider another potential menthor
        menthor++;

  return numPairs;
}

int maxPairingGreedyWeakestMenthorsWeakestStudentsQuadratic(int rating[], int n)  {
  // number of pupil-menthor pairs
  int numPairs = 0;
  // indicates if the person already became a pupil
  int isPupil[MAX];

  for (int i = 0; i < n; i++)
    isPupil[i] = 0;

  // we assign pupils to menthors, starting from the weakest menthors
  // each menthor is assigned the weakest possible unassigned pupil
  for (int menthor = 1; menthor < n; menthor++)
```

```
    for (int pupil = 0; pupil < menthor; pupil++)
      if (!isPupil[pupil] && rating[menthor] >= 2 * rating[pupil]) {
    // we found a pupil for the current menthor
        numPairs++;
        isPupil[pupil] = 1;
        break;
      }
  return numPairs;
}



int maxPairingGreedyStrongestMenthorsStrongestPupils(int rating[], int n)  {
  // number of pupil-menthor pairs
  int numPairs = 0;
  // indicates if the person already became a pupil
  int isPupil[MAX];

  for (int i = 0; i < n; i++)
    isPupil[i] = 0;

  // we assign pupils to menthors, starting from the strongest menthors
  // each menthor is assigned a strongest possible unassigned pupil
  for (int menthor = n-1, pupil = n-2; pupil >= 0; menthor--)
    while (pupil >= 0)
      if (!isPupil[pupil] && rating[menthor] >= 2 * rating[pupil]) {
        // we found a pupil for the current menthor
        numPairs++;
        isPupil[pupil] = 1;
        break;
      } else
    // we consider another potential pupil
    pupil--;

  return numPairs;
}


int maxPairingGreedyStrongestMenthorsStrongestPupilsQuadratic(int rating[], int n)  {
  // number of pupil-menthor pairs
  int numPairs = 0;
  // indicates if the person already became a pupil
  int isPupil[MAX];

  for (int i = 0; i < n; i++)
    isPupil[i] = 0;
```

```
  // we assign pupils to menthors, starting from the strongest menthors
  // each menthor is assigned a strongest possible unassigned pupil
  for (int menthor = n-1; menthor > 0; menthor--)
    // we seek a pupil for the current menthor by checking all candidates
    // this is suboptimal and can lead to quadratic complexity
    for (int pupil = menthor-1; pupil >= 0; pupil--)
      if (!isPupil[pupil] && rating[menthor] >= 2 * rating[pupil]) {
        // we found a pupil for the current menthor
        numPairs++;
        isPupil[pupil] = 1;
        break;
      }

  return numPairs;
}

// all potential pairings are checked - backtracking search
int maxPairingBruteForce_(int rating[], int n, int menthor, int isPupil[], int numPairs) {
  if (menthor == n)
    return numPairs;

  int max = 0;
  for (int pupil = 0; pupil < menthor; pupil++) {
    if (!isPupil[pupil] && rating[menthor] >= 2*rating[pupil]) {
      isPupil[pupil] = 1;
      int num = maxPairingBruteForce_(rating, n, menthor+1, isPupil, numPairs+1);
      if (num > max)
    max = num;
      isPupil[pupil] = 0;
    }
  }

  int num = maxPairingBruteForce_(rating, n, menthor+1, isPupil, numPairs);
  if (num > max)
    max = num;

  return max;
}

int maxPairingBruteForce(int rating[], int n) {
  int isPupil[MAX];
  for (int i = 0; i < n; i++)
    isPupil[i] = 0;
  return maxPairingBruteForce_(rating, n, 0, isPupil, 0);
}
```

```
#ifdef __TESTING__

#include <iostream>
#include <algorithm>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
  srand(time(NULL));
  int rating[MAX];
  int n = rand() % (MAX - 1) + 1;
  for (int i = 0; i < n; i++)
    rating[i] = rand() % 100;
  sort(rating, rating+n);
  cout << maxPairingGreedyWeakestPupilsWeakestMenthors(rating, n) << endl;
  cout << maxPairingGreedyStrongestMenthorsStrongestPupils(rating, n) << endl;
  cout << maxPairingGreedyStrongestMenthorsStrongestPupilsQuadratic(rating, n) << endl;
  cout << maxPairingGreedyWeakestMenthorsWeakestStudentsQuadratic(rating, n) << endl;
  cout << maxPairingBruteForce(rating, n) << endl;
  return 0;
}

#endif
```

Longest palindromic substring

- Write a program that finds the length of the longest substring of the given string that is palindromic (reads the same backwards and forwards).
- longesPalindromicSubstring.cpp

```
int isPalindrome(char s[], int i, int len) {
  int l = i, r = i + len - 1;
  while (l < r) {
    if (s[l++] != s[r--])
      return 0;
  }
  return 1;
}

int longestPalindromicSubstringBruteForce(char s[], int n) {
  for (int len = n; len >= 1; len--)
    for (int i = 0; i + len - 1 < n; i++)
```

```
      if (isPalindrome(s, i, len))
    return len;
  return 0;
}


int longestPalindromicSubstringAnalyzeCenters(char s[], int n) {
  int maxLength = 0;
  for (int i = 0; i < n; i++) {
    int length;

    int k = 1;
    while (i - k >= 0 && i + k < n && s[i - k] == s[i + k])
      k++;
    length = 2 * k - 1;

    if (length > maxLength)
      maxLength = length;

    k = 0;
    while(i - k >= 0 && i + k + 1 < n && s[i - k] == s[i + k + 1])
      k++;
    length = 2 * k;

    if (length > maxLength)
      maxLength = length;
  }

  return maxLength;
}

int longestPalindromicSubstringManacher(char s[], int n) {
  int N = 2 * n + 1;
  int length[MAX];

  int C = 0, R = 0; // L = C - (R - C)
  for (int i = 0; i < N; i++) {
    int i_sym = C - (i - C);
    if (i < R && i + length[i_sym] < R)
      length[i] = length[i_sym];
    else {
      length[i] = i <= R ? R - i : 0;
      if ((i + length[i]) % 2 == 1)
    length[i]++;
      while (i - length[i] - 1 >= 0 && i + length[i] + 1 < N &&
         s[(i - length[i] - 1) / 2] == s[(i + length[i] + 1) / 2])
```

```
    length[i] += 2;
    }

    if (i + length[i] > R) {
      C = i;
      R = i + length[i];
    }
  }

  int maxLength = 0;
  for (int i = 0; i < N; i++)
    if (length[i] > maxLength)
      maxLength = length[i];

  return maxLength;
}

#ifdef __TESTING__

#include <iostream>
using namespace std;

int main() {
  char s[] = "accabcbaabccbaabcbacbcbb";
  int n = sizeof(s) / sizeof(char);
  cout << longestPalindromicSubstringBruteForce(s, n) << endl;
  cout << longestPalindromicSubstringAnalyzeCenters(s, n) << endl;
  cout << longestPalindromicSubstringManacher(s, n) << endl;
  return 0;
}

#endif
```

Prefix-suffix

- prefixSuffix.cpp
- Write a program that calculates the length of the longest proper prefix of a given string that is both its suffix. For example, for the string `abccbaccbaccababccbac` it is the string `abccbac`.

```
int prefixSuffixBruteForce(char str[], int n) {
  int maxD = 0;
  for (int d = 1; d < n; d++) {
    int prefix_suffix = 1;
```

```
    for (int i = 0, j = n - d; i < d; i++, j++)
      if (str[i] != str[j]) {
    prefix_suffix = 0;
    break;
      }
    if (prefix_suffix)
      maxD = d;
  }

  return maxD;
}


int prefixSuffixKMP1(char str[], int n) {
  int kmp[MAX + 1];
  kmp[0] = kmp[1] = 0;
  for (int i = 1; i < n; i++) {
    int k = kmp[i];
    while (1) {
      if (str[i] == str[k]) {
    kmp[i + 1] = k + 1; break;
      } else if (k == 0) {
    kmp[i + 1] = 0; break;
      } else
    k = kmp[k];
    }
  }
  return kmp[n];
}

int prefixSuffixKMP2(char str[], int n) {
  int kmp[MAX + 1];
  kmp[0] = -1;
  for (int i = 0; i < n; i++) {
    int k = i;
    while (k > 0 && str[i] != str[kmp[k]])
      k = kmp[k];
    kmp[i + 1] = kmp[k] + 1;
  }
  return kmp[n];
}
```