Kolokwium ze wstępu do informatyki, I rok Mat.

(*Ściśle tajne* przed godz. 10 : 15 12 grudnia 2005.)

1. Program w C[1] zawiera deklaracje

```
void P1 (int *a, int* b) {
  int i;
  for (i=0;i<10;i++)  a[i] = b[9-i];
}

void P2 (int *a, int  b[]) {
  int i;
  for (i=0;i<10;i++)  a[i] = b[9-i];
}

void P3 (int *a, int* b) {
  int i;
  for (i=1;i<10;i++)  b[i] = a[i-1];
}
```

Jaka będzie zawartość tablicy `c` po wykonaniu instrukcji `P1 ( c, c )`, `P2 ( c, c )` oraz `P3 ( c, c )`, jeśli przed wykonaniem każdej z tych instrukcji tablica ta zawiera liczby $1,\ldots,10$, tj. `c[i-1] = i` dla `i = 1,...,10`?

Rozwiązanie:

Procedury P1 i P2 tylko pozornie różnią się sposobem przekazania tablicy b do procedury. Przekazując tablicę jako parametr podprogramu, przekazujemy w istocie wskaźnik (adres) pierwszego elementu. Dlatego w obu przypadkach efekt wywołania będzie taki sam.

Aby zabezpieczyć się przed przypadkową modyfikacją zawartości tablicy, można zastosować słowo kluczowe const.

```
#include<stdio.h>

void P1 (int *a, int* b) {
  int i;
  for (i=0;i<10;i++)  a[i] = b[9-i];
}

void P2 (int *a, int  b[]) {
  int i;
  for (i=0;i<10;i++)  a[i] = b[9-i];
}

void P3 (int *a, int* b) {
  int i;
  for (i=1;i<10;i++)  b[i] = a[i-1];
}

void wypisz(const int *c ) {
  int i;
  for ( i=0; i<10; i++) printf("%d ", c[i]);
}

void inicjuj(int *c){
  int i;
  for (i=0; i<10; i++) c[i] = i+1;
}

int main () {
  int c[9];

  inicjuj(c); printf("P1: "); P1(c,c); wypisz(c);
  inicjuj(c); printf("P2: "); P2(c,c); wypisz(c);
  inicjuj(c); printf("P3: "); P3(c,c); wypisz(c);

  return 0;
}

/* Wynik:
P1: 10 9 8 7 6 6 7 8 9 10
P2: 10 9 8 7 6 6 7 8 9 10
P3: 1 1 1 1 1 1 1 1 1 1
*/
```

2. Tablica a na pozycjach od 0 do 9 zawiera liczby od 1 do 10 uporządkowane rosnąco. Liczby w tej tablicy zostały następnie posortowane algorytmem HeapSort (priorytet każdej liczby jest równy tej liczbie). Ile przestawień było wykonanych? Podaj zawartość tablicy po każdym przestawieniu.

   Rozwiązanie:

   Opis algorytmu można znaleźć w skrypcie do wykładu. Zamieszczony niżej program *nie jest* wymaganą częścią rozwiązania, ale może być użyty do sprawdzenia rozwiązania podanego dalej.

```
#include<stdio.h>
/* zamiana elementow */
void Swap(int * a, int * b){
  int pom;
  pom = *a;
  *a = *b;
  *b = pom;
}/*Swap*/
```

[1] W roku akademickim 2005/2006 językiem wykładowym był Pascal. Tu zadania z kolokwium zostały przetłumaczone na język C.

```c
/* procedura przywraca porzadek w kopcu */
/* f - pozycja elementu, ktory nie jest na swoim miejscu */
/* l indeks ostatniego elementu w tablicy */
void DownHeap(int tab[], int f,int l) {
  /* indeksy korzeni poddrzew wierzcholka f */
  int i,j,k;
  i = 2*f + 1;
  while (i <=l){
    j = i+1;
    if (j <= l )
      if (tab[j] > tab[i]) i =j;

    if (tab[i] > tab[f]) {
      Swap( &tab[i], &tab[f]);
      for ( k=0; k<10; k++) printf("%d ", tab[k]); printf("\n");
      f= i;
      i = 2*f+1;
    }/*if*/
    else break;
  }/*while*/
}/*DownHeap*/

void HeapSort ( int tab[], int n ) {
  int i,k;
  /*tworzenie kopca*/
  printf("Tworzenie kopca:\n");
  for (i = n/2 -1; i>=0;i--)
    DownHeap(tab,i,n-1);
  /*sortowanie "skopcowanej" tablicy*/
  printf("Sortowanie:\n");
  for (i = n-1;i>0;i--) {
    /*kopiec zawiera element o najwiekszym priorytecie na pozycji 0 w
    tablicy*/
    Swap (&tab[0],&tab[i]);
    /*ustalamy pierwszy element na docelowej pozycji*/
    printf("Swap: "); for ( k=0; k<10; k++) printf("%d ", tab[k]);
    printf("\n");
    /*porzadkujemy mniejszy kopiec*/
    DownHeap(tab,0,i-1);
  }
}/*HeapSort*/

/*Testuejmy rozwiazanie*/
int main(){
  int c[10];
  int i;
  for (i=0; i<10; i++) c[i] = i+1;
  HeapSort ( c, 10 ) ;
  return 0;
}
```

```
/*
Wynik dzialania programu

Tworzenie kopca:
1 2 3 4 10 6 7 8 9 5
1 2 3 9 10 6 7 8 4 5
1 2 7 9 10 6 3 8 4 5
1 10 7 9 2 6 3 8 4 5
1 10 7 9 5 6 3 8 4 2
10 1 7 9 5 6 3 8 4 2
10 9 7 1 5 6 3 8 4 2
10 9 7 8 5 6 3 1 4 2
Sortowanie:
Swap: 2 9 7 8 5 6 3 1 4 10
9 2 7 8 5 6 3 1 4 10
9 8 7 2 5 6 3 1 4 10
9 8 7 4 5 6 3 1 2 10
Swap: 2 8 7 4 5 6 3 1 9 10
8 2 7 4 5 6 3 1 9 10
8 5 7 4 2 6 3 1 9 10
Swap: 1 5 7 4 2 6 3 8 9 10
7 5 1 4 2 6 3 8 9 10
7 5 6 4 2 1 3 8 9 10
Swap: 3 5 6 4 2 1 7 8 9 10
6 5 3 4 2 1 7 8 9 10
Swap: 1 5 3 4 2 6 7 8 9 10
5 1 3 4 2 6 7 8 9 10
5 4 3 1 2 6 7 8 9 10
Swap: 2 4 3 1 5 6 7 8 9 10
4 2 3 1 5 6 7 8 9 10
Swap: 1 2 3 4 5 6 7 8 9 10
3 2 1 4 5 6 7 8 9 10
Swap: 1 2 3 4 5 6 7 8 9 10
2 1 3 4 5 6 7 8 9 10
Swap: 1 2 3 4 5 6 7 8 9 10
*/
```

Całkowita liczba przestawień sklada się z 8 przestawień podczas budowy kopca, 9 przestawień na miejsca docelowe w algorytmie HeapSort (oznaczone Swap) oraz 13 przestawień porządkujacych kopiec. Razem 30 przestawień.

3. Algorytm znajdowania liczb pierwszych został zrealizowany w następujący sposób:

```
1  #define n 1000 /*rozmiar tablicy*/
2
3  void Sito(int * tab) {
4    int i, j;
5    for (i=2;i<n;i++) tab[i] = 1;
6
7    for (i=2; i < n/2;i++) {
8      j = i+i;
9      while (j < n ) {
10       tab[j] = 0;
11       j += i;
12     }/*while*/
13   }/*for*/
14 }/*Sito*/
```

Wskaż numer linii z instrukcją, którą można uznać za operację dominującą w tym algorytmie. Oblicz jego koszt. Wiedząc, że $\sum_{i=2}^{n} \frac{1}{i} < \ln n$ dla $n \geq 2$, oszacuj rząd złożoności tego algorytmu w zależności od n.

Rozwiązanie:

Wyznaczanie kosztu można sprowadzić do liczenia tzw. operacji dominujących. Operacje dominujące to takie działania w algorytmie, którym można przyporządkować co najwyżej pewną ustaloną liczbę pozostałych operacji.

Operacją dominującą jest przypisanie w lini 10.

Algorytm znajdowania liczb pierwszych dla każdej liczby $2 \leq d \leq \frac{n}{2}$ (pętla for) wykona $\frac{n}{2d}$ operacji (pętla while).

W sumie będzie to $\frac{n}{2} * (\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{\frac{n}{2}})$. Wiedząc, że $\sum_{i=2}^{n} \frac{1}{i} < \ln n$ otrzymujemy:

$$\frac{n}{2} * \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{\frac{n}{2}}\right) \leq \frac{n}{2} \ln \frac{n}{2}.$$

Rząd złożoności algorytmu jest równy $n \log n$.

4. Tablica a, indeksowana od 0 do $n-1$ ($n > 0$) zawiera ciąg n liczb uporządkowany niemalejąco. Napisz podprogram w C, który dla ustalonego x (podanego jako parametr) znajduje w takiej tablicy liczbę `a[i]` taką, że $|x - a[i]| \leq |x - a[j]|$ dla `j=1,...,n`. Indeks `i` ma być wynikiem. Zrealizowany algorytm ma być możliwie szybki, ale przede wszystkim poprawny i dobrze objaśniony (za pomocą komentarzy w kodzie lub dodatkowego opisu). Podaj złożoność tego algorytmu.

Można użyć standardowej w C funkcji `abs` z biblioteki `<stdlib.h>`, która oblicza wartość bezwzględną parametru. Typem tej funkcji jest `int`.

Rozwiązanie:

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int liniowe(int n, int x, const int *a) {
6    int i, min, result =0;
7      /* sprawdzamy po kolei wszystko */
8    min = abs(a[0]-x);
9    for (i=1;i<n;i++) {
10     if (abs(a[i]-x)<min) result = i;
11   }/*for*/
12   return result;
13 }/*liniowe*/
14
15 /*optymalne - wskazowka rozwiazania*/
16 int logartymiczne(int n, int x, const int *a) {
17   int l, r, p;
18
19     /* osobno sprawdzamy, czy x nie jest miedzy pierwszym i ostatnim, */
20     /* tj. najmniejszym i najwiekszym */
21   if ( x <= a[0] ) return 0;
22   else if ( x >= a[n-1] ) return n-1;
23     /* jesli jest, to wykonujemy wyszukiwanie binarne */
24   l = 0;  r = n-1;
25   while ( r-l > 1 ) {
26     p = (r+1)/2;
27     (x <= a[p]) ? r = p : l = p;
28   }/*while*/
29   if ( l == r )
30     return l;
31   else
32     return (abs(a[l]-x) < abs(a[r]-x )) l ? r;
33 }/*logarytmiczne*/
```

Kolokwium ze Wstępu do Informatyki, I rok Mat.

(*Ściśle tajne* przed godz. 12 : 15 10 grudnia 2007.)

Proszę bardzo uważnie przeczytać treść zadań. Na ocenę bardzo duży wpływ będzie miała czytelność rozwiązań i poprawność uzasadnienia każdej odpowiedzi.

1. Podany niżej podprogram w C:

```
1  void SelectionSort ( int n, float tab[] ) {
2    int i, j, k;
3    float a;
4
5    for ( i = n-1; i > 0; i-- ) {
6      for ( k = i, j = k-1; j >= 0; j-- )
7        if ( tab[j] > tab[k] ) k = j;
8      a = tab[k];
9      tab[k] = tab[i];
10     tab[i] = a;
11   }
12 } /*SelectionSort*/
```

realizuje tzw. algorytm sortowania przez wybieranie.

Wskaż operację dominującą w tym algorytmie i oblicz jego złożoność pesymistyczną i optymistyczną w zależności od długości sortowanego ciągu.

Co można powiedzieć o złożoności średniej tego algorytmu?

Rozwiązanie:

Operacją dominującą jest porównanie w lini 7 procedury SelectionSort.

Szukamy w zbiorze elementu największego i wymieniamy go z elementem na ostatniej pozycji. W ten sposób element najmniejszy znajdzie się na swojej docelowej pozycji. W identyczny sposób postępujemy z resztą elementów należących do zbioru. Znów wyszukujemy element największy i zamieniamy go z elementem na przedostatniej pozycji. Otrzymamy dwa posortowane elementy. Procedurę kontynuujemy dla pozostałych elementów dotąd, aż wszystkie będą posortowane.

W zagnieżdzonej pętli for mamy kolejno $n-1, n-2, \ldots, 1$ porównań.

$$(n-1)+(n-2)+\ldots+1 = \frac{n\cdot(n-1)}{2}$$

Algorytm zawsze wykona taką sama liczbę operacji dominujących, więc złożoność pesymistyczna, optymistyczna oraz średnia są sobie równe. Rząd tych wszystkich złożoności jest równy n^2.

2. W tablicy tab znajduje się początkowo następujący ciąg liczb:

$$12, 1, 11, 2, 10, 3, 9, 4, 8, 5, 7, 6$$

Przyjmujemy, że priorytetem liczby jest ona sama. Rozważamy dwa sposoby uporządkowania kopca zbudowanego z tych liczb:

a) `for ( i = 1; i < 12; i++ ) UpHeap ( tab, i );`

b) `for ( i = 5; i >= 0; i-- ) DownHeap ( tab, i, 11 );`

Dla każdego z tych sposobów podaj wynik, czyli narysuj uporządkowany kopiec lub napisz wynikowy ciąg elementów w tablicy, oraz podaj liczbę wykonanych operacji porównywania priorytetów.

Rozwiązanie:

Opis algorytmu można znaleźć w skrypcie do wykładu. Zamieszczony program *nie jest* częścią rozwiązania, ale może być użyty do jego sprawdzenia.

```
1  #include<stdio.h>
2  /* zamiana elementow*/
3  void Swap(int * a, int * b){
4    int pom;
5    pom = *a;
6    *a = *b;
7    *b = pom;
8  }/*Swap*/
9
10 /*procedura przywraca porzadek w kopcu*/
11 /*f - pozycja elementu, ktory nie jest na swoim miejscu*/
12 /*l indeks ostatniego elementu w tablicy*/
13 void DownHeap(int tab[], int f,int l) {
14   /*indeksy korzeni poddrzew wierzcholka f*/
15   int i,j,k;
16   i = 2*f + 1;
17   while (i <=l){
18     j = i+1;
19     if (j <= l )
20       if (tab[j] > tab[i]) {
21         i =j; for ( k=0; k<12; k++) printf("%d ", tab[k]); printf("\n");
22       }/*if*/
23     if (tab[i] > tab[f]) {
24       Swap( &tab[i], &tab[f]);
25       for ( k=0; k<12; k++) printf("%d ", tab[k]); printf("\n");
26       f= i;
27       i = 2*f+1;
28     }/*if*/
29     else break;
30   }/*while*/
31 }/*DownHeap*/
```

```c

/*procedura przywraca porzadek w kopcu*/
/*n indeks ostatniego elementu w tablicy*/
void UpHeap(int tab[], int n) {
  /*i - indeks wierzcholka "wyzej", k - licznik petli for*/
  int i,k;
  i = (n-1)/2;
  /*pniemy sie w "gore" z konca do indeksu 0*/
  while (n>0){
    /*porownanie priorytetow*/
    if (tab[n] > tab[i]) {
      Swap( &tab[n], &tab[i]);
      for ( k=0; k<12; k++) printf("%d ", tab[k]); printf("\n");
      n = i;
    }/*if*/
    else break;
  }/*while*/
}/*UpHeap*/

/*Testujemy rozwiazanie*/
int main(){
  int tab[12] = {12,1,11,2,10,3,9,4,8,5,7,6};
  int i;
  printf("UpHeap:\n");
  for (i=1; i<12; i++)   UpHeap(tab,i);
  printf("DownHeap:\n");
  int tab2[12] = {12,1,11,2,10,3,9,4,8,5,7,6};
  for (i=5; i >= 0; i--) DownHeap(tab2,i,11);

return 0;
}


/*
Wynik dzialania programu

UpHeap:
12 2 11 1 10 3 9 4 8 5 7 6
12 10 11 1 2 3 9 4 8 5 7 6
12 10 11 4 2 3 9 1 8 5 7 6
12 10 11 8 2 3 9 1 4 5 7 6
12 10 11 8 5 3 9 1 4 2 7 6
12 10 11 8 7 3 9 1 4 2 5 6
12 10 11 8 7 6 9 1 4 2 5 3
DownHeap:
12 1 11 2 10 6 9 4 8 5 7 3
12 1 11 2 10 6 9 4 8 5 7 3
12 1 11 2 10 6 9 4 8 5 7 3
12 1 11 8 10 6 9 4 2 5 7 3
12 1 11 8 10 6 9 4 2 5 7 3
12 1 11 8 10 6 9 4 2 5 7 3
12 10 11 8 1 6 9 4 2 5 7 3
12 10 11 8 1 6 9 4 2 5 7 3
12 10 11 8 7 6 9 4 2 5 1 3
*/
```

3. Napisz w języku C (ale czytelnie!) podprogram, którego pierwsze dwa parametry opisują ciąg $a_0, \ldots, a_{n-1}$ liczb typu `float` (pierwszy parametr typu `int` określa długość ciągu, drugi jest tablicą), a następne dwa parametry to wskaźnik `m` zmiennej typu `int` i tablica `b` zmiennych typu `int`.

   Zadaniem podprogramu jest znalezienie wszystkich nie dających się wydłużyć podciągów $a_k, \ldots, a_l$, złożonych z kolejnych elementów ciągu $a_0, \ldots, a_{n-1}$, i takich że ciąg iloczynów $a_k, a_k \cdot a_{k+1}, \ldots, a_k \cdot \ldots \cdot a_l$ jest ściśle rosnący. Dla każdego takiego podciągu należy do tablicy `b` wpisać (na kolejnych pozycjach) liczby `k` i `l`. Zmiennej wskazywanej przez parametr `m` należy przypisać liczbę znalezionych podciągów.

   Należy założyć, że tablica `b` jest dostatecznie długa.

   Uwaga: liczby a_i nie muszą być dodatnie.

   Rozwiązanie:

```c
#include<stdio.h>
#define size_b 1000 /* zakladamy, ze tablica b jest dostatecznie dluga*/

/*Parametry funkcji zgodnie ze specyfikacja zadania*/
void Podciagi(int n, const float * a, int * m, int* b) {
  int i;              /*licznik w tabeli a*/
  int j;              /*licznik konca rosnacego podciagu*/
  int k = 0;          /*licznik w tabeli b*/
  float pom=0;        /*zmienna pomocnicza*/
  /*petla po elementach z tabeli a*/
  for (i =0 ; i<n-1;i++) {
    /* czy mamy poczatek ciagu rosnacego?*/
    if (a[i] < a[i]*a[i+1]) {
      /*wpisujemy indeks poczatku ciagu rosnacego*/
      b[k] = i; k += 2;
      /*zwiekszamy liczbe podciagow rosnacych*/
      *m = *m +1;

      /*czy mozemy go przedluzyc?*/
      pom = a[i]*a[i+1]; j = i + 2;
      /*sprawdzamy, jak dlugi jest ciag rosnacy */
      /* (uwaga na koniec tablicy a)*/
      while (pom < pom*a[j] && j < n) {
        pom *= a[j];
```

```
        j++;
      }/*while*/
      /*wpisujemy indeks konca ciagu rosnacego*/
      b[k-1] = j-1;
    }/*if*/
  }/*for*/
}/*Podciagi*/

/*Procedure Podciagi mozemy przetestowac w programie*/
int main()
{
  int n=5; /*rozmiar tablicy tab*/
  float tab[5] = {-1,-1,-1,-1,-2};
  int m = 0, j,l;
  int b[size_d];
  for ( j=0; j<size_d; j++)
    b[j] = 0;
  Podciagi(n, tab, &m, b);
  for ( l=0; l<size_d; l++)
    printf("%d ", b[l]);
  printf("\n");
}
/*Wynik dzialania programu: 0 1 1 2 2 3 3 4 0 0 */
```