1. Написать программу

Код программы

**var**

M,Rez,Ind:**array of** integer;

i,s,n:integer;

**Procedure** MergeSort(left,right:integer);

**var**

middle,j,i,n1,n2:integer;

**begin**

**if** right<=left **then**

**exit**

**else**

**begin**

middle:=(left+right) **div** 2;

MergeSort(left,middle);MergeSort(middle+1,right);

n1:=left; n2:=middle+1;

**for** i:=left **to** right **do**

**begin**

**if** (n1 < (middle + 1))**and**((n2 > right)**or**(M[Ind[n1]] < M[Ind[n2]])) **then**

**begin**

Rez[i]:=Ind[n1];

inc(n1);

**end**

**else**

**begin**

Rez[i]:=Ind[n2];

inc(n2);

**end**;

**end**;

**for** i:= left **to** right **do**

Ind[i]:=Rez[i];

**end**;

**end**;

**procedure** Search(p,b,e:integer; **var** M,Ind:**array of** integer);

**var** midl:integer;

**begin**

**while** b < e **do**

**begin**

midl:=(b + e) **div** 2;

**if** M[Ind[midl]] < p **then**

b := midl + 1

**else**

e:=midl;

**end**;

**if** M[Ind[b]]=p **then**

writeln(Ind[b])

**else**

writeln('Элемент не найден');

**end**;

**begin**

write('Введите количество эелментов: ');

readln(n);

SetLength(M,n);

SetLength(Rez,n);

SetLength(Ind,n);

**for** i:=0 **to** n-1 **do begin**

M[i]:=Random(11);

Ind[i]:=i;

**end**;

writeln('Отсортированный массив');

MergeSort(0, n-1);

**for** i:=0 **to** n-1 **do**

write(M[Ind[i]]:3);

writeln();

write('Введите элемент для поиска: ');

readln(s);

write('Результат поиска: ');

Search(s,0,n-1,M,Ind);

**end**.

Трудоемкость

Процедура слияния: O(n) (будет H(n) = 2n)

Процедура сортировки слиянием: O(n\*logn)

(Процедура слияния с линейной трудоемкостью

выполнится log2N раз

(из-за рекурсивного вызова для половин массива)

1. Каковы трудоемкости этих процедур

По инверсии: если X[i]>X[j], при I < j.

Максимум инверсий:

Если j=2, тогда 1

Если j=3, тогда 2

Пусть {X[1] <= X [2] <= … <= X[i]}, X[i] > X[i+1]

Всего инверсий: 1 + 2 + … + (n-1) = n \* (n-1)/2

• Чтобы устранить все инверсии нужно: 1\*(1+1)/2+2(2+1)/2+…+(n-1)\*n/2=~ n^3/6

• В худшем случае трудоемкость равна O(n^3)

2)

Трудоемкость считается n\*n действий, потому что обрабатывается два цикла.

При вычислении получится: 1 + 2 +…+ (n-1)=n(n-1)/2 => В худшем случае трудоемкость O(n^2)

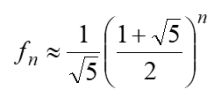
3)

А) **Нерекурсивное вычисление НОД**

Наихудший случай: если при fk>=max(a,b) > fk-1

Где fk , fk-1 - числа фибоначчи, выполняется условие: max(a,b) = fk , min(a,b) = fk-1

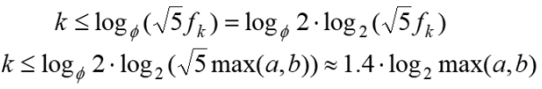
Тогда k - кол-во выполнений цикла. Пренебрегая вторым слагаемым в формуле Муавра:



Б) **Рекурсивное вычисление НОД**

Трудоемкость: log(max(a, b))

И логарифмируя левую и правую части, получим:



Трудоемкость составляет O(log max(a,b))

4)

**procedure** hanoi(n, a, b, c: integer);

**begin**

**if** n = 1 **then**

writeln(a, ' > ', c)

**else**

**begin**

hanoi(n - 1, a, c, b);

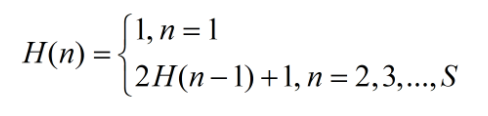
writeln(a, ' > ', c);

hanoi(n - 1, b, a, c);

**end**;

**end**;

Рекуррентное соотношение:



Из рекуррентного соотношения следует:

H(n)=2H(n-1)+1=22H(n-2)+2+1=2n-1 +…+2+1=2n-1

В худшем случае трудоемкость O(n^3)