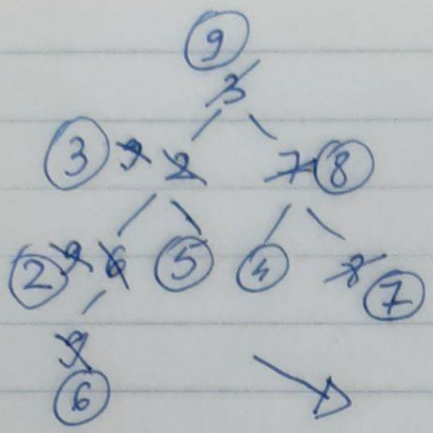
Heap (Arbore de selecție / arbore parțial ordonat)

este un AB aproape complet unde informația din nodul tată și fii este într-o relație de ordine.
Pentru Heap maxim: cheia tatălui este mai mare sau egală cu a fiilor.
Pentru Heap minim: similar

Înălțimea unui ansamblu Heap cu n noduri este log2n-1.
Reprezentarea cu ajutorul unui vector unde v[i] este tată pentru elementele v[2*i] și v[2*i+1].
Rădăcina arborelui este v[1].
Algoritmii implementați pentru un ansamblu Heap (cu respectarea proprietății de ansamblu Heap la orice pas):

1. **crearea ansamblului**
2. **inserarea unui nod**
3. **extragerea unui nod**

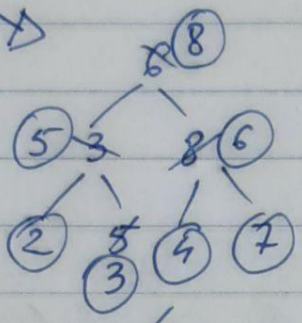
max heap



3 2 7 6 5 4 8 9

9 3 8 2 5 4 7 6

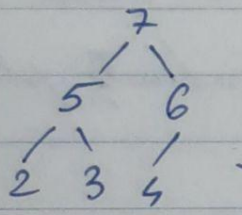
6 3 8 2 5 4 7 | 9

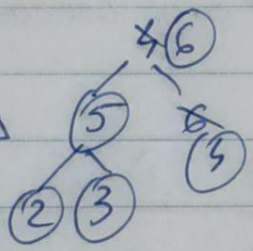


8 5 6 2 3 4 7 | 9

7 5 6 2 3 4 | 8 9

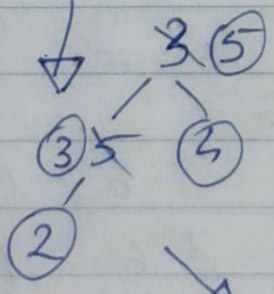
4 5 6 2 3 | 7 8 9





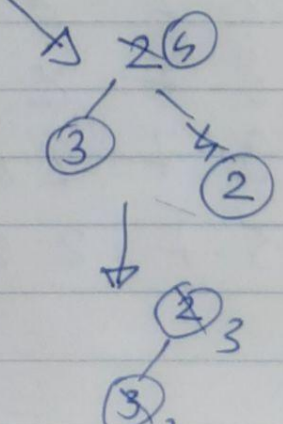
6 5 4 2 3 | 7 8 9

3 5 4 2 | 6 7 8 9



5 3 4 2 | 6 7 8 9

2 3 4 | 5 6 7 8 9



4 3 2 | 5 6 7 8 9

2 3 | 4 5 6 7 8 9

3 2 4 5 6 7 8 9

2 | 3 4 5 6 7 8 9

1. crearea ansamblului:

Obs:
j - var pentru specificare numărul de elemente puse în vector;
i - var pentru indicele nodului curent;

```
void creare(int &n, int v[])
{ int i,j,aux;
  cin>>n;
  cin>>v[1];
  for(j=2;j<=n;j++)
  {
     cin>>v[j]; i=j;
     while(i>1)
           if(v[i]>=v[i/2])
              aux=v[i],v[i]=v[i/2],v[i/2]=aux,i=i/2;
           else
                i=1;
  }
}
```

2. inserarea unui nod:

```
void inser(int x,int &n, int v[])
{ v[++n]=x;
  int i=n;
  while(i>1)
     if(v[i]>=v[i/2])
           x=v[i],v[i]=v[i/2],v[i/2]=x,i/=2;
     else i=1;

 }
```

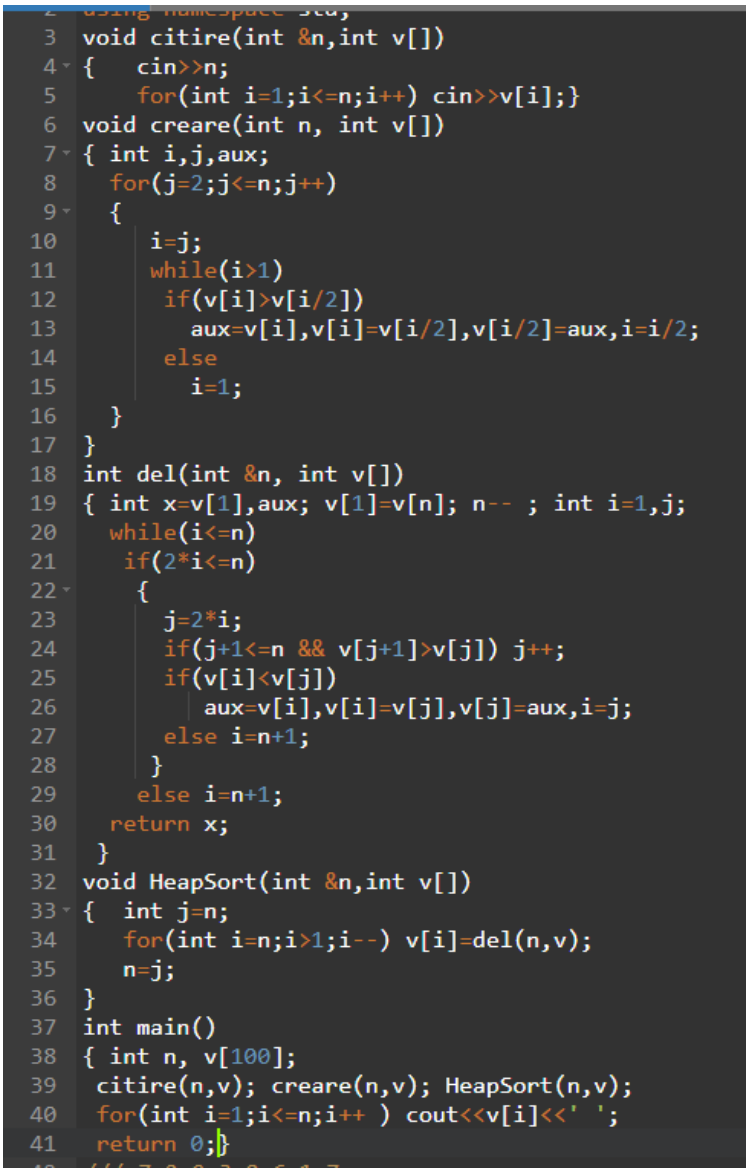
3. stergerea unui nod:

```
int del(int &n, int v[])
{ int x=v[1]; v[1]=v[n]; n - - ; int i=1,j;
  while(i<=n)
   if(2*i<=n)
    {
      j=2*i;
      if(j+1<=n && v[j+1]>=v[j]) j++;
      if(v[i]<=v[j])
               aux=v[i],v[i]=v[j],v[j]=aux,i=j;
      else i=n+1;
     }
    else i=n+1;
```

```
  return x;
 }
```

HeapSort

```
 void HeapSort(int &n,int v[])
{  int j=n;
    for(int i=n;i>1;i--) v[i]=del(n,v);
    n=j;
}
main()
{
 creare(n,v);
 HeapSort(n,v);
 // afisarea vectorului
}
```



```
3  void citire(int &n,int v[])
4  {   cin>>n;
5      for(int i=1;i<=n;i++) cin>>v[i];}
6  void creare(int n, int v[])
7  { int i,j,aux;
8    for(j=2;j<=n;j++)
9    {
10      i=j;
11      while(i>1)
12       if(v[i]>v[i/2])
13         aux=v[i],v[i]=v[i/2],v[i/2]=aux,i=i/2;
14       else
15         i=1;
16   }
17 }
18 int del(int &n, int v[])
19 { int x=v[1],aux; v[1]=v[n]; n-- ; int i=1,j;
20   while(i<=n)
21    if(2*i<=n)
22     {
23       j=2*i;
24       if(j+1<=n && v[j+1]>v[j]) j++;
25       if(v[i]<v[j])
26          aux=v[i],v[i]=v[j],v[j]=aux,i=j;
27       else i=n+1;
28      }
29     else i=n+1;
30   return x;
31  }
32 void HeapSort(int &n,int v[])
33 {  int j=n;
34    for(int i=n;i>1;i--) v[i]=del(n,v);
35    n=j;
36 }
37 int main()
38 { int n, v[100];
39  citire(n,v); creare(n,v); HeapSort(n,v);
40  for(int i=1;i<=n;i++ ) cout<<v[i]<<' ';
41  return 0;}
```

```
#include <iostream>
using namespace std;
void printv(int v[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << v[i] << " ";
    cout << "\n";
}
void heapi(int v[], int n, int i)
{
    printv(arr,9);
    int mav = i, st = 2 * i + 1, dr = 2 * i + 2;
    if (st < n && v[st] > v[mav]) mav = st;
    if (dr < n && v[dr] > v[mav]) mav = dr;
    if (mav != i) { swap(v[i], v[mav]); heapi(v, n, mav); }
}
void heapSort(int v[], int n)
{
     for (int i = n / 2 - 1; i >= 0; i--)
        heapi(v, n, i);
    for (int i = n - 1; i > 0; i--)
    {
        swap(v[0], v[i]);
        heapi(v, i, 0);
    }
}
int main()
{
    int v[] = { 12, 11, 13, 5, 6, 7 , 18, 9, 2};
    int n = sizeof(v) / sizeof(int);
    heapSort(v, n);
    printv(v, n);
}
```