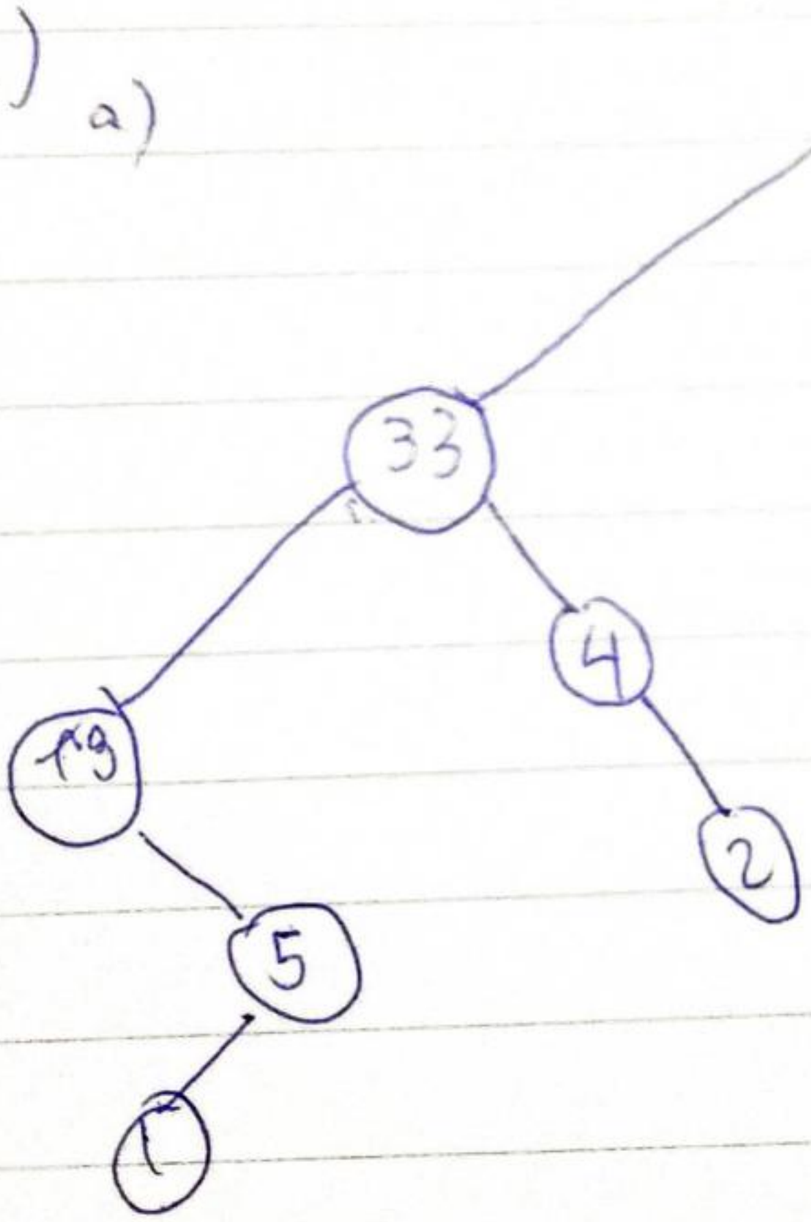
NICOLAE ALEXANDRU

GOODYEAR
MADE TO FEEL GOOD.

1) Se poate face prin parcurgere, ținând o variabilă de tip maxim unde stocăm valoarea maximă

```
maxim ← -99999
pentru x ← i,j execută
    dacă maxim < V[x] atunci
        maxim ← V[x]
```

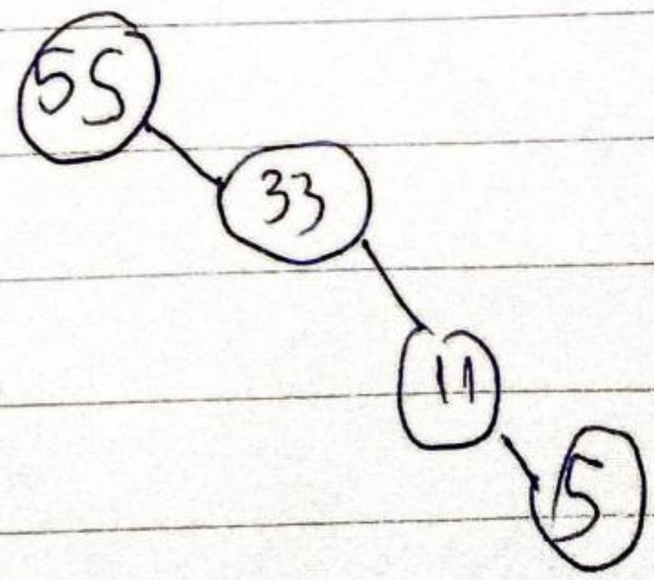
2) a)



b) Vectorul trebuie să fie descrescător

Exemplu

V[] = { 55, 33, 11, 5 }



NICOI ALEXANDRU

Nicoi

c) ~~pentru~~ stack s

~~s -> push (a[0])~~
~~left[0] = -1~~
~~pentru i <- 1, n-1~~
~~cât timp (a[i] < s->top)~~
~~s -> pop~~
~~dacă~~

```
stack s
  s-> push (<a[0], 0>)
  pentru i <- 1, n-1
     cât timp (a[i] < s->top și s->empty != true)
        s->pop
     dacă s->empty = true
        left[i] = -1
        s->push (<a[i], i>)
     altfel
        left[i] = s->top->i
        s->push (<a[i], i>)

s->clear
s->push (<a[n-1], n-1>)
pentru i <- n-2, 0, -1
   cât timp (a[i] > s->top și s->empty != true)
      s->pop
   dacă s->empty = true
      right[i] = -1
      s->push (<a[i], i>)
   altfel right[i] = s->top->i
      s->push (<a[i], i>)
```

GOODYEAR
MADE TO FEEL GOOD.

Nicoi' ALEXANDRU

1) a)

3) Un max-heap este un arbore binar complet în care valoarea stocată în orice nod este mai mare sau egală decât valoarea fiilor săi

b) Algoritmul de mai sus are o complexitate de O(n log n) din cauza inserării și ștergerii în heap log n de n ori, chiar dacă crearea arborelui cartezian în O(n)

Algoritmul este optim dacă vectorul ar fi aproape sortat, însă dpdv al spațiului necesită memorie suplimentară pentru heap. (în plus încă O(n)).

2 d)

```
root <- 0, last <- 0
pentru i <- 1, n-1
    last <- i-1
    right[i] <- -1
    cât timp ( a[last] <= a[i] și last ≠ root)
        last <- tata[last]
    dacă ( a[last] <= a[i])
        left[root] = i
        left[i] = root
        root = i
```

NICOI' ALEXANDRU


```
altfel daca  right[last] = -1
             right[last] = i
             data[i] = last
             left[i] = -1
altfel
    data[right[last]] = i
    left[i] = right[last]
    right[last] = i
    data[i] = last
```