CS5800: Algorithms — Virgil Pavlu

Homework 2

Name: Mumuksha Pant
Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. (Extra Credit) Implement MergeSort without recursive calls.

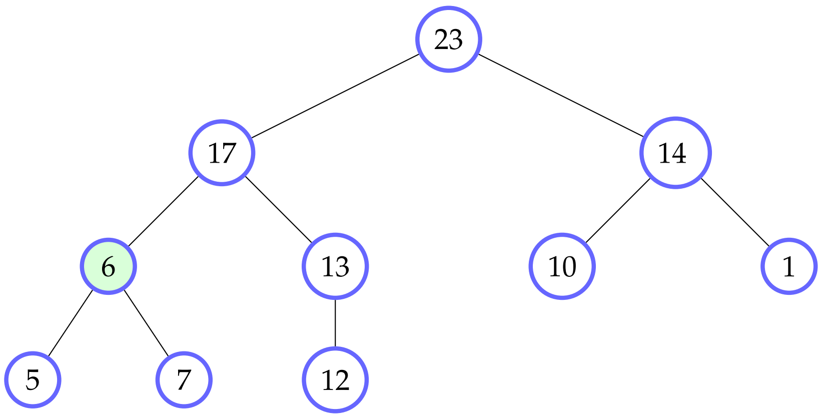
2. (5 points) Exercise 6.1-4, explain why.

Solution:
For every Max-heap there exists a property that value of each node (i) is **AT MOST** the value of its parent, i.e, $A[Parent(i)] \geq A[i]$.
In other words, the max element will become the root node. So the smallest elements would reside in the leaf nodes that is the nodes at the end of the tree which don't have any children.

---

3. (5 points) Exercise 6.1-6, explain why.

Solution:
A= [23, 17, 14, 6, 13, 10, 1, 5, 7, 12]
NO . As per max-heap property : $A[Parent(i)] \geq A[i]$
Since the parent A[3] is smaller than its child A[8], it violates max heap property.



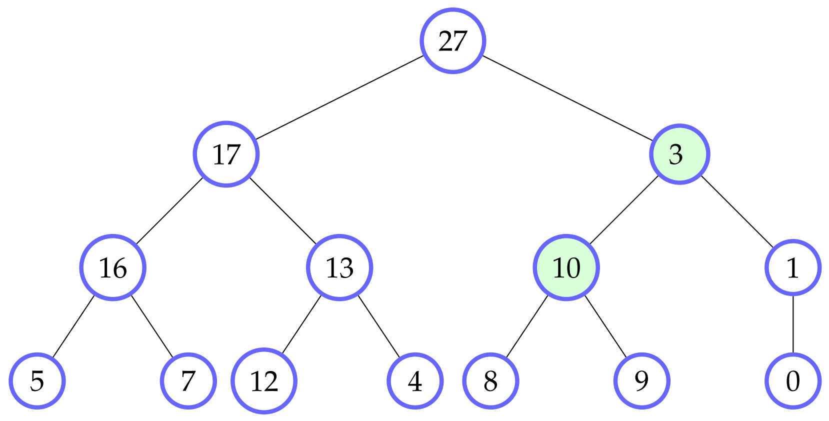
---

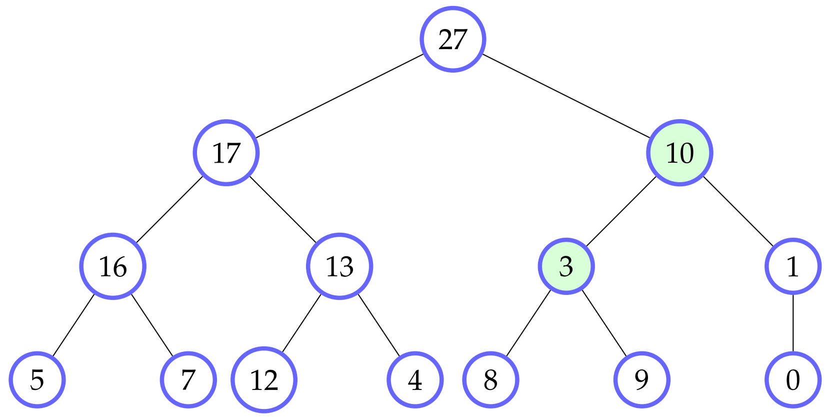
4. (10 points) Exercise 6.2-1.

Solution:
A[27,17,3,16,13,10,1,5,7,12,4,8,9,0]
Element 3 is wrongly placed, violating the max-heap property

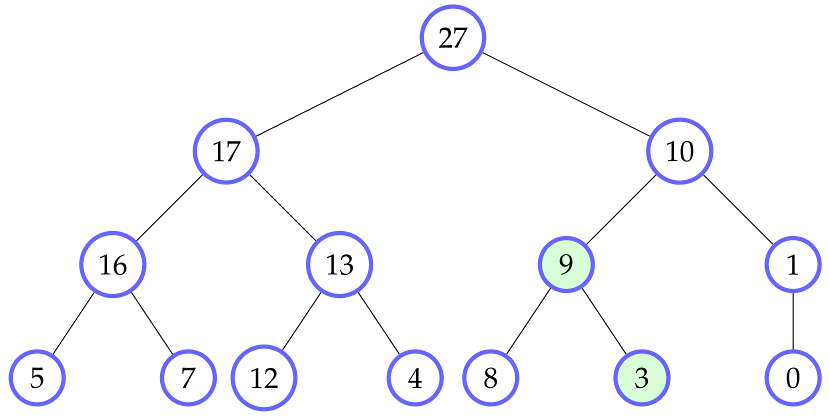
a) To Max Heapify (A,3) :

swap (A[3], A[6])
swap elements 3 and 10





b) To Max Heapify (A,6) swap (A[6], A[13])
swap elements 9 and 3



Solution : A=[27, 17, 10, 16, 13, 9, 1, 5, 7, 12, 4, 8, 3, 0]

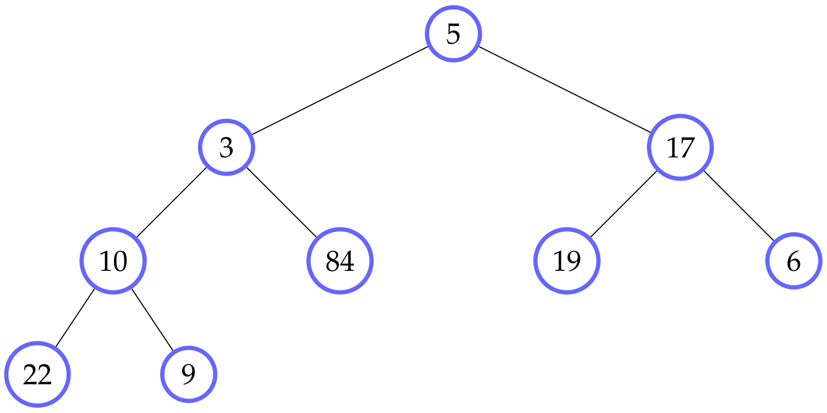
---

5. (10 points) Exercise 6.3-1.

Solution:
BUILD-MAX-HEAP (A)

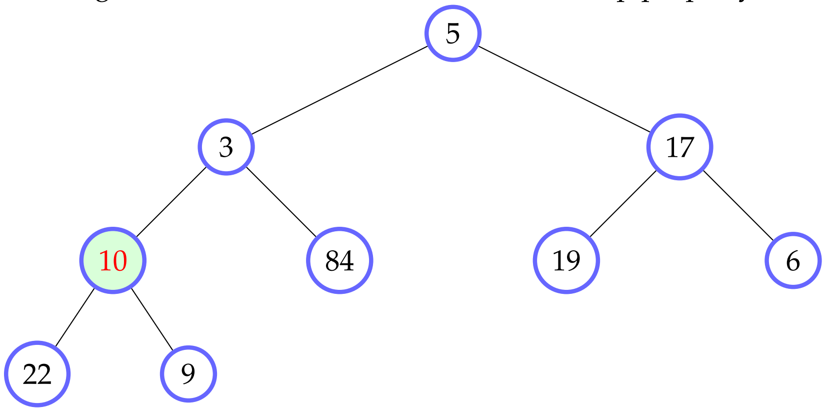
```
1.  A.heapSize = A.length
2.  for i = [A.length/2] downto 1
3.      MAX-HEAPIFY(A,i)
```

Given array:
A = [5; 3; 17; 10; 84; 19; 6; 22; 9]



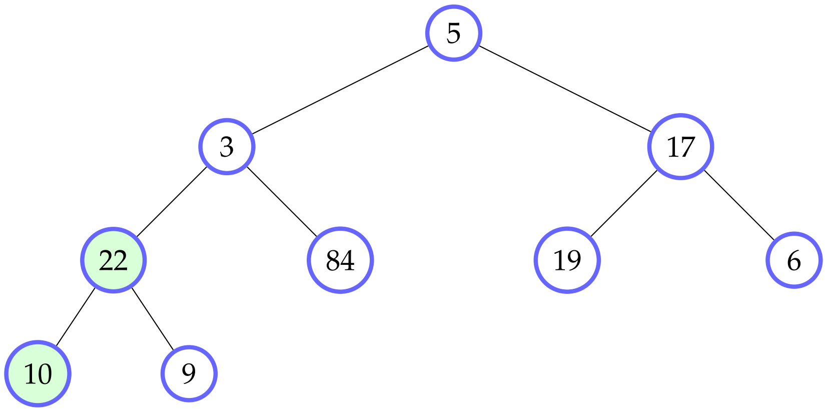
$\frac{Array.length}{2} = \frac{9}{2} = 4$

we go to 4th element and check the max heap property.

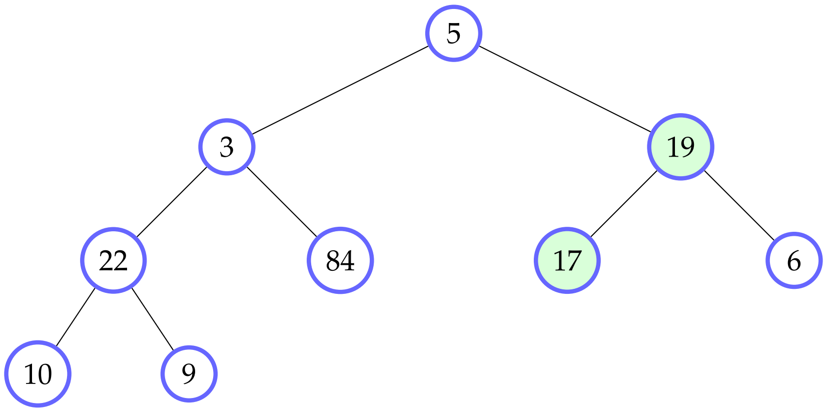


It does not satisfy max- heap property. Therefore we need to fix this

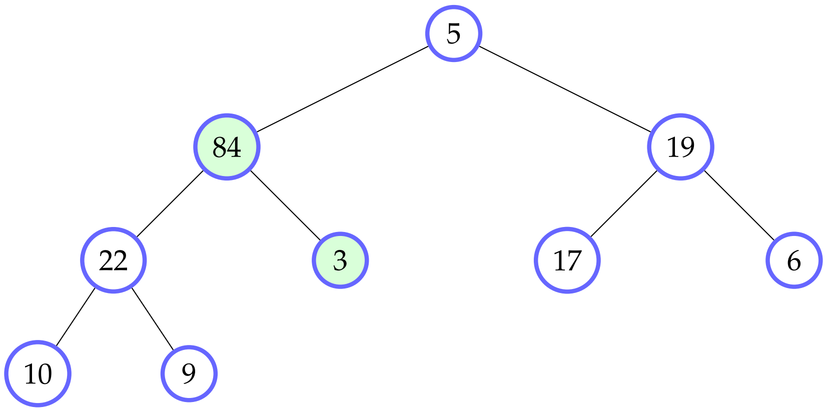
1. MAX-HEAPIFY (A , 4)
swap (A[4] , A[8])



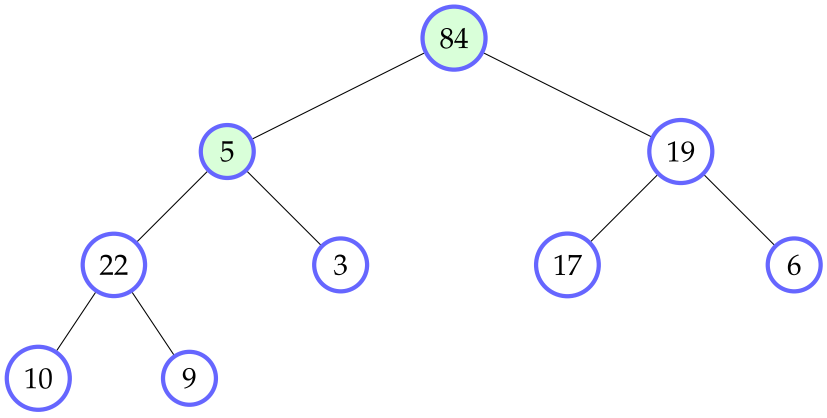
2.MAX-HEAPIFY (A , 3)
swap (A[3] , A[6])



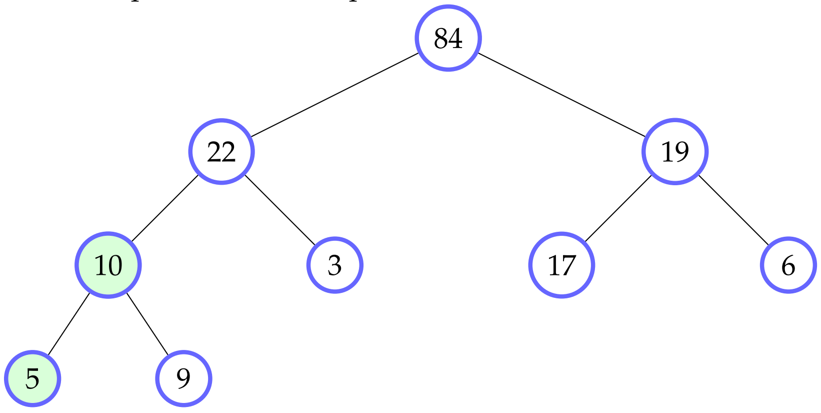
3.MAX-HEAPIFY (A , 2)
swap (A[2] , A[5])



4.MAX-HEAPIFY (A , 1)
swap (A[1] , A[2])



5. swap (22,5) then swap (10,5)

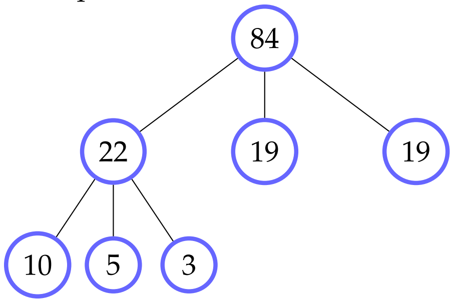


Solution : $\boxed{\text{A= [84, 22, 19, 10, 3, 17, 6, 5, 9]}}$

---

6. (15 points) Problem 6-2.

Solution:
a. For a $d-ary$ heap :
root would be i
parent[i] = i/d
j^{th} child = $d(i+j)$ where $j = 0$ till $d-1$
left child : di
right child : d(i+1)
Example - Assume d=3,



Array representation - [84,22,19,19,10,5,3]

b.

Computing height for d

nodes, height

0	0
1	d
2	d^2
3	d^3
4	d^4
..	...

calculating the height : $n = d^{k-1}$

taking log both sides :

$\log_d n = \log_d d^{k-1}$

$\log_d n = k - 1$

so height of a d-ary tree is $\boxed{log_d n}$

c)

EXTRACT-MAX sorts the value of the maximum element, moving the minimum element to the root of the heap, and then calls ”max-heapify” to maintain heap property.

EXTRACT MAX

```
1.  $if heap.size < 1$
        error “heap underflow”

2.  MAX = A[1] //root is the max
3.  A[1] = A[n]   // n = A.heap-size
4.  n =n-1
5.  MAX - HEAPIFY ( A, 1, n , d )
6.  return MAX
```

Calculating Run time :

height of d-ary heap : $log_d n$

maximum children in each node : d

$\boxed{\theta(d * \log_d n)}$

d. **MAX-HEAP-INSERT (A,key)**

```
1. A.heap-size= A.heap.size+1
2. A[A.heap-size] =-infinity
3. HEAP-INCREASE-KEY ( A, A.heap.size, key )
A[i]=key
while $i > 1$ AND $A[Parent(i)] < A[i]$
swap A[i] , A[Parent (i)]
```

i=Parent(i) ;

Calculating Run time :
using the above algorithm, we have to traverse the height of the heap to perform max-heapify
height of d-ary heap : $log_d n$
$\boxed{\theta(\log_d n)}$

e. **Increase-key (A,A.heap.size,key)**

```
1.  A[i]=key
2.  while $i > 1$ AND $A[Parent(i)] < A[i]$
3.          swap A[i] , A[Parent (i)]
4.            i=Parent(i) ;
```

Running Time:
run time is same as traversing the height of the heap :
$\boxed{\theta(\log_d n)}$

---

7. (5 points) Exercise 7.2-1.

Solution:
$T(n) = T(n-1) + \theta(n)$ ..k=1
$T(n) = T(n-2) + (n-1) + \theta(n)$ ..k=2
$T(n) = T(n-3) + (n-2) + (n-1) + \theta(n)$ ..k=3
$T(n) = T(n-k) + n.k[(n-1) + (n-2) + ...(n-k)] + \theta(n)$ ..k
$T(n) = T(n-k) + n.k \sum_1^k (n-k) + \theta(n)$ ..k

using formula for sum of n natural numbers:
Finding Last k : put (n-k) =0 ; n=k
T(0) = T(n-k) =0
$T(n) = T(n-k) + n.k - [\frac{k(k+1)}{2}] + \theta(n)$

$T(n) = T(0) + n.k - [\frac{k(k+1)}{2}] + \theta(n)$
put n=k ;
$T(n) = T(0) + n^2 - [\frac{n(n+1)}{2}] + \theta(n)$
$T(n) = 0 + n^2 - [\frac{n^2}{2} - \frac{n}{2}] + \theta(n)$
$T(n) = \frac{n^2}{2} - \frac{n}{2} + \theta(n)$

$T(n) = \theta(n^2) recursivepart + \theta(n) nonrecursivepart$

---

8. (5 points) Exercise 7.2-2, explain why.

Solution:

A =

2	2	2	2	2	2	2

highlighted element is pivot

QUICKSORT (A ,p, r)

```
1.   $if p < r$
       then $q- > PARTITION(A,p,r)$
2.   Quicksort (A, p, q-1)
3.   Quicksort (A, q+1, r)
```

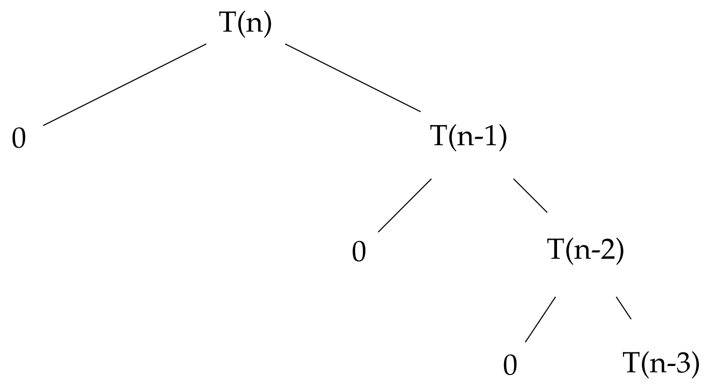
PARTITION

```
....
1.  FOR
2.      if A[j] ¡= pivot
3.          i=i+1;
4.          swap A[i], A[j]
5. End FOR
```

After running the algorithm, when we divide this array,
$Quicksort(A,p,q-1)- > Quicksort(A,0,5)$
$Quicksort(A,q+1,r)- > Quicksort(A,6,6)$
q becomes equal to r

This implies, parition of elements will happen at the last position of Array and the algorithm will show **worst case complexity** behavior.

Running time calculation -



$T(n) = c * T(n-k)$
$T(n) = c * n(n+1)/2$
$T(n) = c * n^2$

$T(n) = \theta(n^2)$

---

9. (5 points) Exercise 7.2-3.

Solution:

A =

10	9	8	6	5	4	3

highlighted element is pivot

QUICKSORT (A ,p, r)

```
1.  $if p < r$
     then $q -> PARTITION(A, p, r)$
2.  Quicksort (A, p, q-1)
3.  Quicksort (A, q+1, r)
```

PARTITION

```
....
1.  FOR
2.      if A[j] ¡= pivot
3.          i=i+1;
4.          swap A[i], A[j]
5. End FOR
```

After running the algorithm, when we divide this array,
in every iteration (partition) , (A[r] = 3) which is taken as the pivot, is the smallest element.
Every partition will produce 2 sub parts (sub arrays - one array containing the smallest element (which was the pivot earlier) and other array containing the remaining elements) .
This will lead to the **Worst Case Complexity** of Quick Sort :

$RunningTime : \theta(n^2)$

---

10. (15 points) Exercise 7.4-1.

Solution:
Collaborator : Dhruv Saini
T(n) = $max_{0<=q<=n-1}$ (T(q) +T(n-q-1)) + $\theta(n)$
has Solution $\Omega(n^2)$

We guess : $c.n^2 \leq T(n)$ where c is a constant

Then , $c.[(T(q^2) + T(n-q-1)^2) + \theta(n)] \leq T(n)$

max value of the above expression will be when q=0

$c.[(T(0) + T(n-0-1)^2) + \theta(n)] \leq T(n)$
$c.[0 + T(n-1)^2) + \theta(n)] \leq T(n)$
$c.[T(n-1)^2] \leq T(n)$
$\Omega(n^2) \leq T(n)$

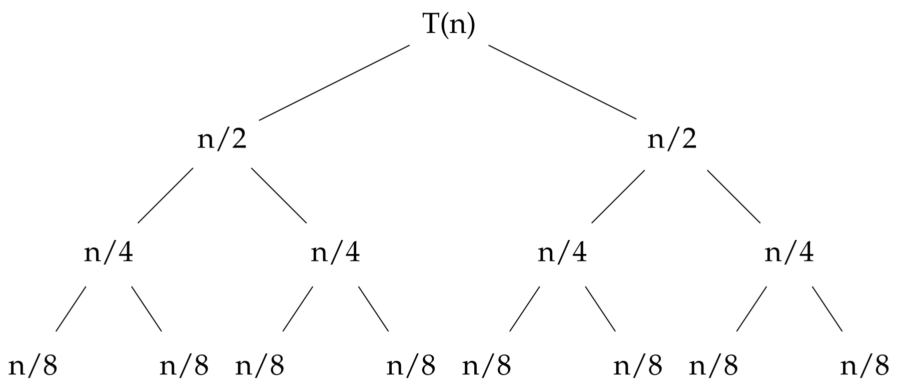
Worst Case has to be "AT LEAST" n^2 because the runtime when pivot is in the extremes is n^2

---

11. (15 points) Exercise 7.4-2.

Solution:
Approach 1 :
Intutively we know , when we have almost equal partitions, then the best case occurs and pivot comes to be the middle element. All subarrays would be the size of approx n/2 .

Running time calculation -



Running Time $T(n) = 2.T(n/2) + c.n$
using Masters Theorem CASE 2 , a=2, b=2 , k=1
$\theta(n\log n)$

Approach 2 :
collaborator : Dhruv Saini + Discussed in Monday Office Hours

how would we know that the best case would be when pivot is in the middle ?
we know that we have a "good" case of n.logn , but it is not necessary that it is the BEST Case
$T(n) = min_{0<=q<=n-1}[T(q) + T(n-1-q) + n] >= c.nlogn$
minimum means the best case, ie, the min value we get from this will be the best case.

Induction Step :

$T(q) >= cqlogq$
$(1)T(n+q) >= c.(n-1-q).log(n-1-q)$
$T(q) >= min[c.q.logq + c(n-1-q).log(n-1-q)] + n$
(2)
using (1) and (2)
$min[c.q.logq + c(n-1-q).log(n-1-q)] + n >= c.qlogq$ -(3)
solving using differentiation -
T'(q) = $c.logq + c - clog(n-q-1) + c(n-q-1)(-1)$

after solving -
log q = log(n-q-1)

q= n-q-1
q=n-1 / 2
Putting this value of q in (3) AND solving it
c.(n-1) log(n-1/2) ¿= c.nlog n

---

12. (10 points) Exercise 7.4-3.

Solution:
$f(x) = x^2 + (n-1-x)^2$
to approach this problem , we have to find max of f(x)
$f(x) = x^2 + (n-1)^2 + x^2 - 2xn + 2x$
$f(x) = 2x^2 + 2(1-n)x + (n-1)^2$

$f'(x) = 4x - 2n + 2$
f"(x) = 4
x = n+1 / 2
this is a parabolic equation and the max of parabola is at extremities when x=0 or x=n-1

---

13. (Extra credit) Problem 6-3.