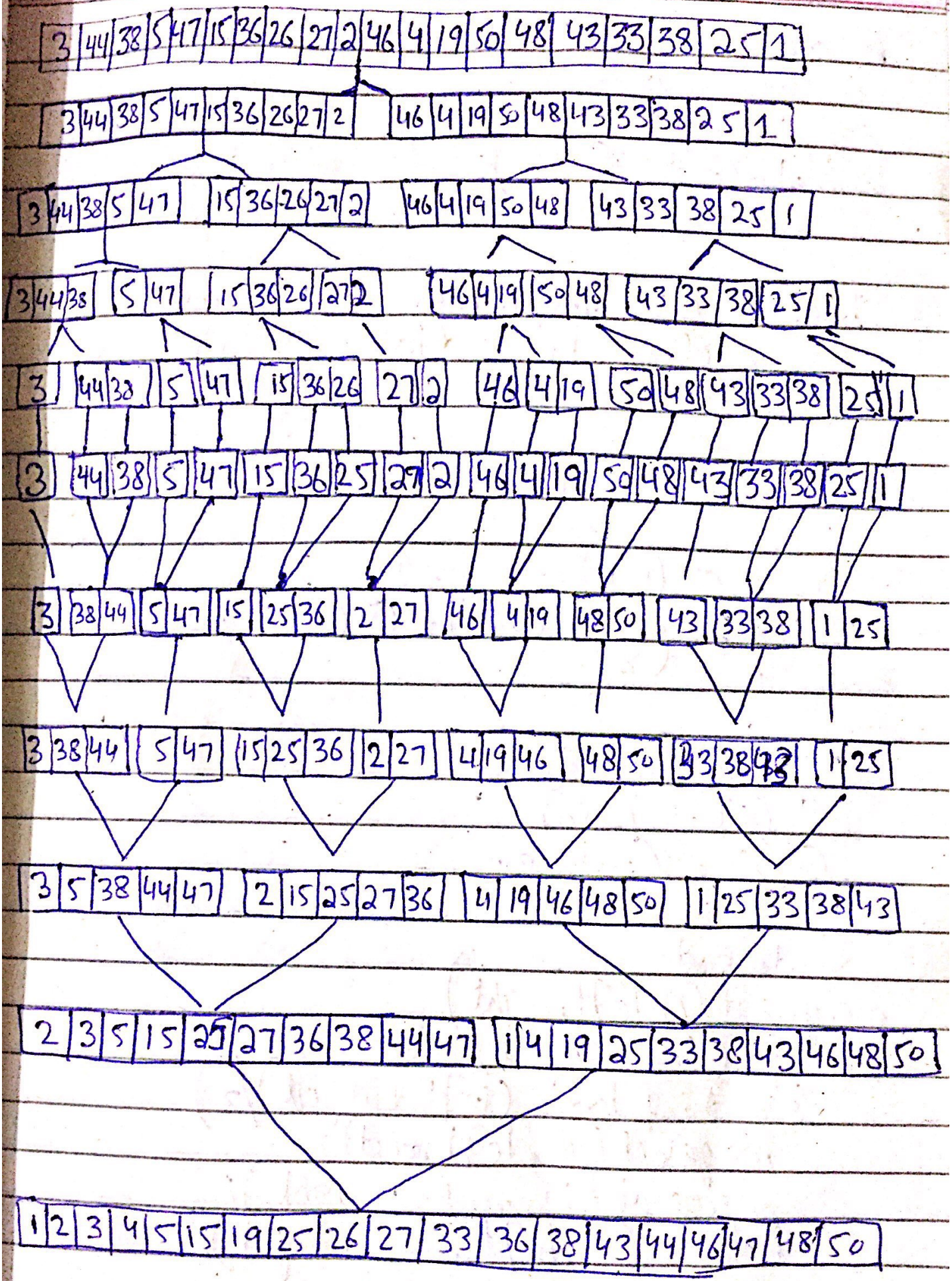
Merge Sort



we are dividing list into 2 parts again and again.

$D(n) = \Theta(1)$

Conquer (combine/merge procedure on n-elements take $\Theta(n)$ time.

So we have

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(1) + \Theta(n) & n > 1 \end{cases}$$

$2T(n/2)$ = Conquer.
$\Theta(1)$ = $D(n)$
$\Theta(n)$ = Combine.

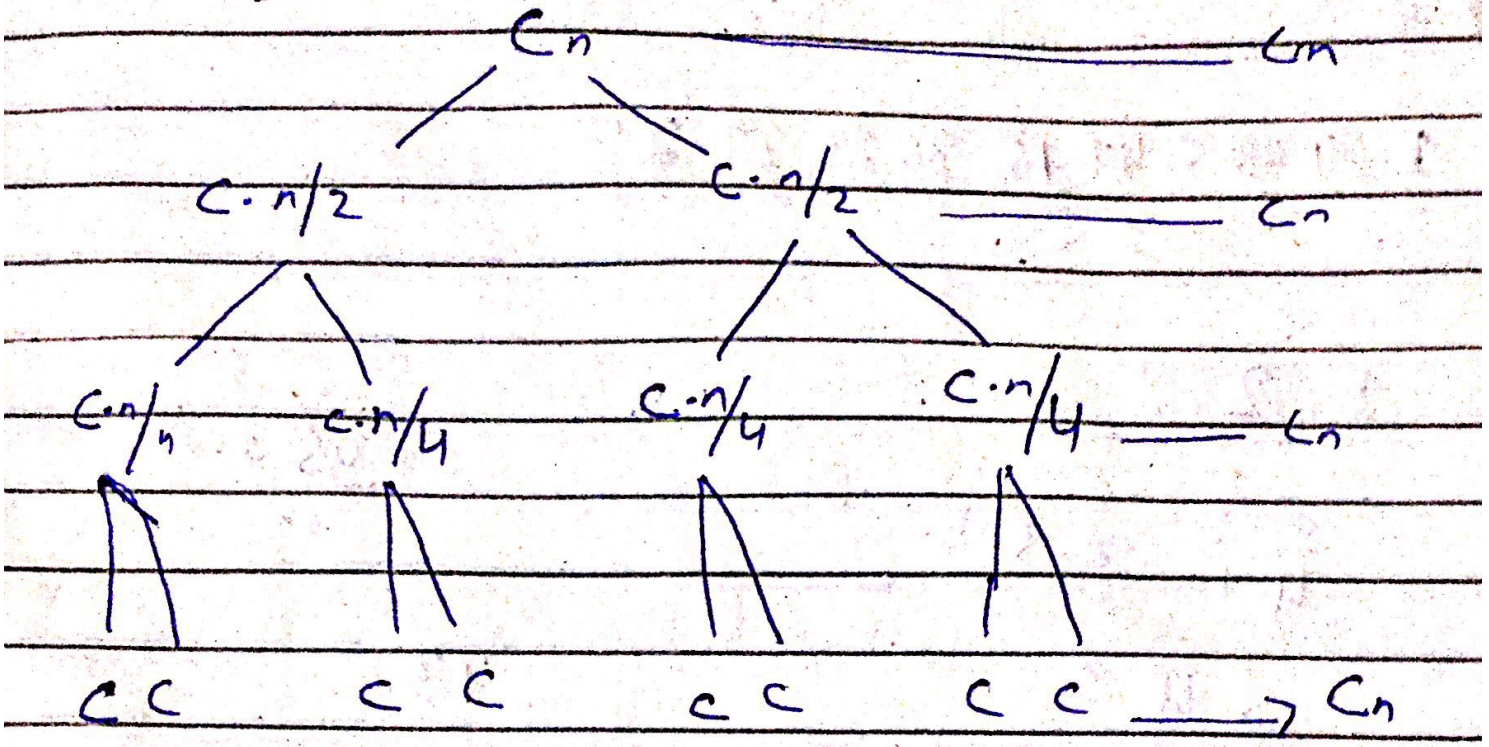
Sudo code

```
mergesort (A, left, right)
{ if (left < right)
  { mid = floor ((left + right)/2)
    mergesort (A, left, mid);
    mergsort (A, mid+1, right);
    Merge (A, left. mid, right); }
}
```

So

c_n = cost of every level

$(\lg_n + 1)$ = no. of levels.



So $c_n(\lg_n + 1)$

$= c_n \lg_n + c_n$

where c is constant and lower order terms c_n are get

$$\Theta(n \cdot \lg n)$$

Quick sort.

3	44	38	5	47	15	36	26	27	2	46	4	19	50	48	43	33	34	25	1

Let Q = 3 → move right → left

2	1	[3]	44	38	5	47	15	36	26	27	46	4	19	50	48	43	33	34	25
p	r	(Q+1)=p																	r

1	2	3	38	5	15	36	26	27	4	19	43	33	34	25	[44]	47	50	48
			p											r		p		r

1	2	3	5	15	36	26	27	4	19	33	34	25	[38]	43	44	[47]	48	50
			p									r						

1	2	3	4	[5]	15	36	26	27	19	33	34	25	38	43	44	47	48	50
					p							r						

1	2	3	4	5	[15]	36	26	27	19	33	34	25	38	43	44	47	48	50
						p						r						

1	2	3	4	5	15	26	27	19	33	34	25	[36]	38	43	44	47	48	50
						p					r							

1	2	3	4	5	15	19	25	[26]	27	33	34	36	38	43	44	47	48	50
							p			r								

1	2	3	4	5	15	19	25	26	27	33	34	36	38	43	44	47	48	50
								p		r								

1	2	3	4	5	15	19	25	26	27	33	34	36	38	43	44	47	48	50

sorted

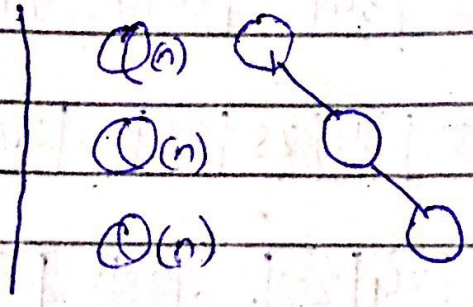
Sudo code

QuickSort(array A, int P, int r)

```
1) if (r > p)
2)   then
3)     i ← a random index from [p .... r]
4)     Swap A[i] with A[p]
5)     q ← partition (A, p, r)
6)     Quicksort (A, p, r-1)
7)     Quick sort (A, q+1, r)
```

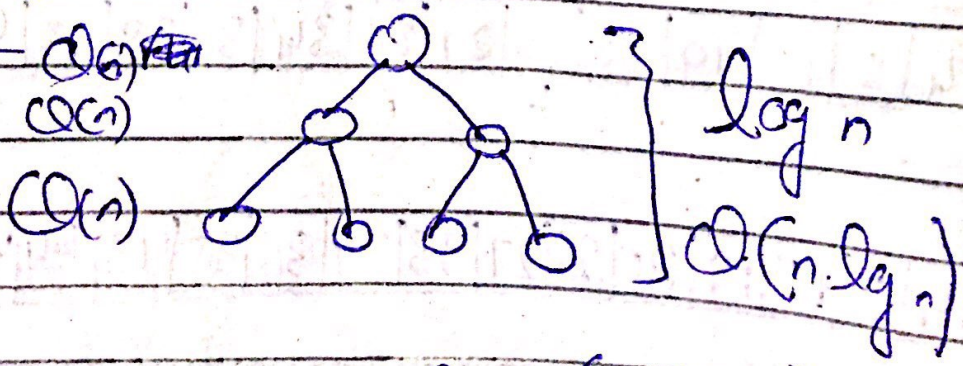
worse case

actually a tree is generated at backend of quick sort.



So running time $\Theta(n^2)$ or $O(n^2)$

Avg case



So running time ~~$\Theta(n)$~~ $(n \cdot \log_2 n)$

Mathematical Prove

At the partion algorithm time is

$$T(n) = T(q-1) + T(n-q) + n$$

(if $q = 1$)

$$T(n) \leq T(0) + T(n-1) + n$$

$$T(n) = T(q-1) + T(n-q) + n$$

$\therefore q = 1$

$$= T(0) + T(n-1) + n$$

$$\boxed{T(n) = T(n-1) + n + 1}$$

(if $q = n$)

$$T(n) = T(n-1) + T(n-n) + n$$

$$T(n) = T(n-1) + T(0) + n$$

$$\boxed{T(n) = T(n-1) + n + 1}$$

So by putting $q = 1$ & $q = n$ we get the same answer for best case worse case.

So
$T(n) = 2c/n(n^2/2 \ln n - n^2/4) + n + 4/n$

$= 2c/n(n^2/2 \ln n - n^2/4) + n + 4/n$

$= cn \ln n - cn/2 + n + 4/n$

$= cn \ln n + n(1 - c/2) + 4/n$

$T(n) = cn \ln n + n(1 - c/2) + 4/n$

to finish proof, we want all of this to be at most $cn \ln n$. For this to happened, we will need to select (c) such that

$$n(1 - c/2) + 4/n \leq 0$$

If $c = 3$

- $n(1 - c/2) + 4/n = n(1 - 3/2) + 4/n$

$= 4/n - n/2$

put $(n = 3)$ $\leq 4/3 - 3/2$

$= -1/6 \leq 0$

So $T(n) = 3 \cdot n \ln n$

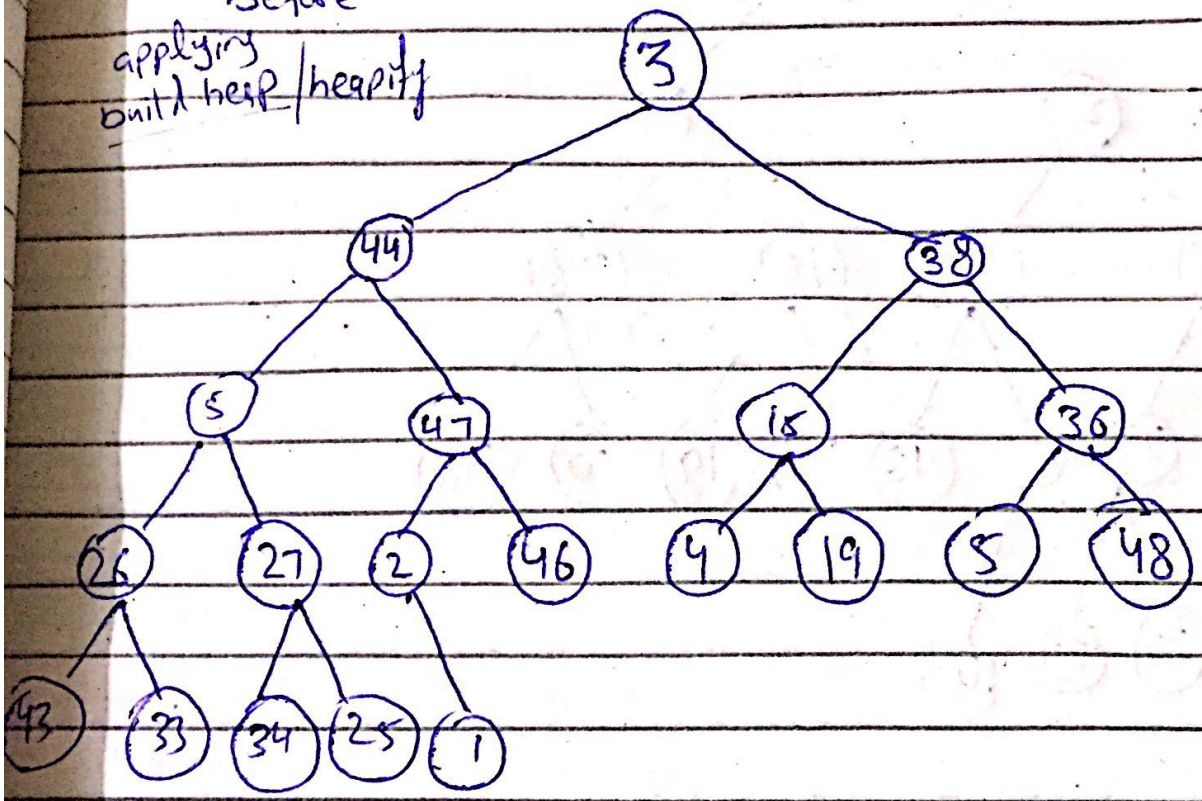
$= \Theta(n \log n)$

HeapSort

Given array

3	44	38	5	47	15	36	26	27	2	46	4	19	50	48	43	33	34	25	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

before applying build heap / heapify

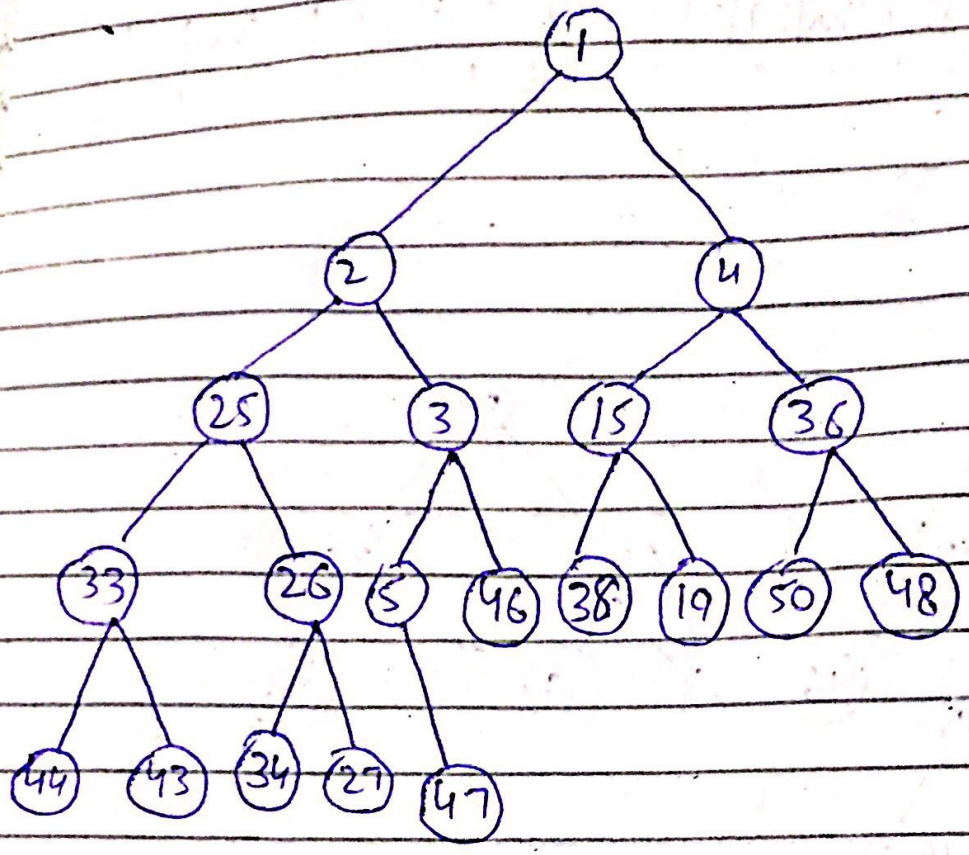


first

We have to know about heapify() and insertion into heap and deletion from heap either from max heap or from min heap. Because running of both (max & min heap) are same.

(min heap)

after apply Build heap.



The values in array become ~~sorted~~

1	2	4	25	3	15	36	33	26	5	46	38	19	50	48	44	43	34	27	47

Now the array is in min-heap formate.

If we start deletion elements one by one from min heap (deletion always start from root in heap).
The data given after deleting element and storing them is in sort form.

1	2	3	4	5	15	19	25	26	27	33	36	38	43	44	46	47	48	50

after deleting and storing element in another array the given array is now in sorted state.

(Analyze heapsort)

i) call to buildheap()
which takes $O(n)$ time

2) Each of $(n-1)$ calls heapify() taken $O(\lg n)$
why $(n-1)$?
=> because at last when there is only 1 node at last then after deletion, no need to check array.
Thus running time of heapify() is

$$\underbrace{O(n)}_{\text{buildheap()}} + \underbrace{(n-1)}_{\text{calls}}\underbrace{O(\lg n)}_{\text{heapify()}}$$

$$O(n) + O(n \lg n)$$

ignore lower order terms —

$$\Rightarrow O(n \lg n)$$