CS-202

ASSIGNMENT - 03

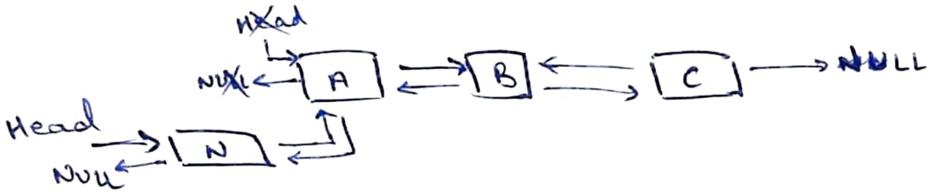
S. SAMARTH REDDY
B19109
GROUP - 25

① These operation can be done using a doubly linked list.

We need to maintain Pointers to the beginning, end and to median location.

(i) Inserting at beginning

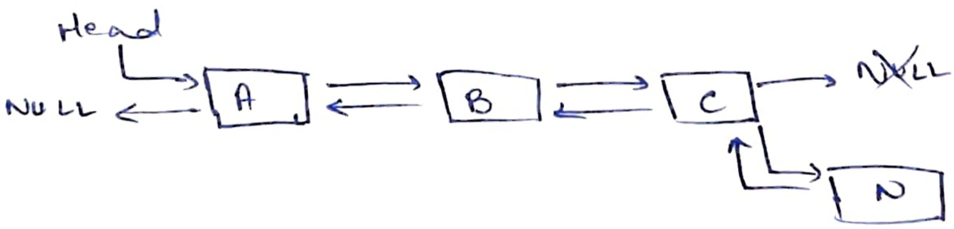
Can be dones in O(1) time as



Change the median Pointer depending on whether n is even or add.

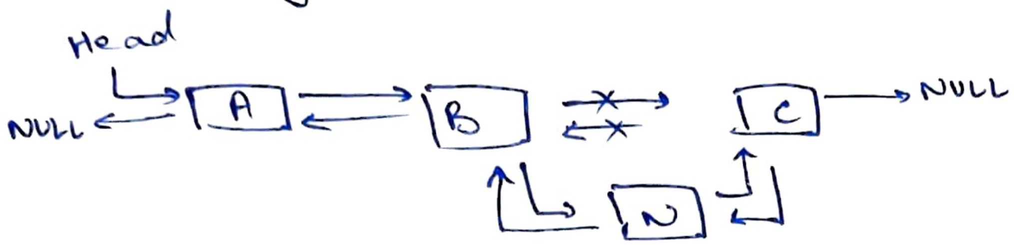
(ii) Inserting at end

This can also be done in O(1) time if we have a Pointer Pointing to the last element.



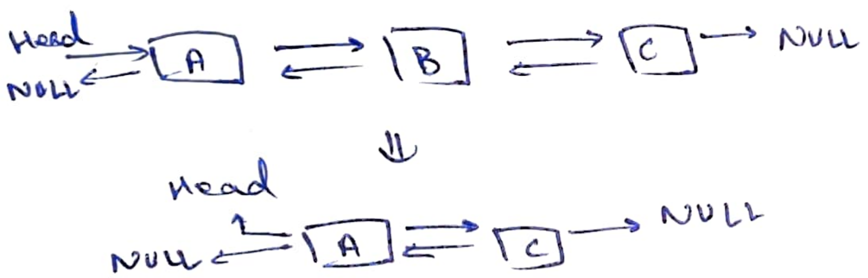
Again Shift median Pointer depending on n

(iii) Inserting at Median



This too can be done in O(1) time, if we have the pointer to the Previous node.

(iv) Delete Second last element



This too can be done in O(1) time, change the median Pointer here too depending upon whether n is odd or even.

(2) The algorithm will work in linear time, if groups of 7 are used.

There are atleast $n/7$ groups with atleast $4 \cdot \lceil 1/2 \cdot \lceil n/7 \rceil \rceil - 2)$ elements that are less than or equal to the median of medians, and at least as many that are greater than or equal to the median of medians. Thus, the larger subset after partitioning has atmost $n - (2n/7 - 8) = 5n/7 + 8$ elements.

- Also, given we can sort 7 elements using 14 comparisions. ∴ Computing medians of each group of 7 takes $14 \times n/7 = 2n$ comparisions.
- Splitting n elements takes $(n-1)$ comparision or $O(n)$ time.

$$\therefore \quad T(n) = T(n/7) + T(8 + 5n/7) + 2n + n$$

$$= T(n/7) + T(8 + 5n/7) + 3n$$

Let's assume $T(n)$ is of form cn [substitution method]

$$T(n) \le c(n/7) + c(8 + 5n/7) + 3n = n(3 + 6c/7) + 8c$$

$$T(n) \le cn, \quad \text{if} \quad cn \ge 3n + \frac{6nc}{7} + 8c$$

$$c\left(\frac{n}{7} - 8\right) \geq 3n$$

$$c \geq \frac{3n}{\frac{n}{7} - 8}$$

$$c \geq \frac{21n}{n - 56}$$

n must be greater than 56,

So if $n > 2 \cdot 56 = 112$, we get $c > 42$

$\therefore$ So for $n > 112$, $\boxed{T(n) < 42n}$

In worst case we can say for $n > 112$,

$T(n) = 42n$.

③ Median of Medians $(A[n], \text{rank}(r))$

(i) Divide the n elements of the input array into $\lceil n/3 \rceil$ groups of 3 elements each and at most one group made up of remaining ($n \bmod 3$) elements

(ii) Find median of each of the $\lceil n/3 \rceil$ groups and add them to set M.

(iii) Perform above step recursively to find the median x of set M.

(iv) Partition of input array into two sets S and L, where S contains elements smaller than x and L elements greater than x.

(v) If $|S| < r-1$ return $(r-|S|+1)^{th}$ smallest element from L.

If $|S| = r-1$ return x

If $|S| > r-1$ return the r^{th} rank element in S

Let $T(n)$ denote the running time of this algorithm.

Finding median of 3 elements of each group can be done in $O(n)$ time.

Spitting the input into two sets can also be done in $O(n)$ time.

In step (iii) we perform the procedure recursively over input of size $n/3$ which would take $T(n/3)$ steps.

Now, there are atleast $2\left(\left\lceil \frac{1}{2}\left\lceil n/3 \right\rceil \right\rceil - 2\right)$ elements which are smaller than x and a like number are greater than x. Thus, in Step 5 size of subproblem is atmost $n - \left(n/3 - 4\right) = \frac{2n}{3} + 4$

$\therefore$ The recurrence relation for the running time becomes $\left[T(n) = T(n/3) + T\left(\frac{2n}{3} + 4\right) + O(n)\right]$

By substitution, it can shown that solution is not of the form cn.

$$cn \geq \frac{2nc}{3} + \frac{nc}{3} + 4c + kn$$

$$0 \geq 4c + kn$$ no constant c exists

$\therefore T(n) \neq O(cn)$

Using the recursion tree method we can show each ~~step~~ level takes $O(n)$ time, and no of levels is of order, $\log n$.

$\therefore T(n) = O(n \log n)$

This can be verified by substitution.

(4) → Construct a min heap with the n elements.

This takes $O(n)$ time.

→ Now, delete the minimum element $(\sqrt{n}-1)$ times from the min-heap. Each delete operation (minima) takes $O(\log n)$ time.

So total time taken for this step $= O((\sqrt{n}-1)\log n)$

→ After deleting the minima $(\sqrt{n}-1)$ times, the minimum element will be the $(\sqrt{n})^{th}$ smallest element of original elements. This minima can be found in $O(1)$ time.

∴ Total time taken to find $(\sqrt{n})^{th}$ smallest element:

$$T(n) = O(n) + O((\sqrt{n}-1)\log n) + O(1)$$

$$\boxed{T(n) = O(n)}$$

[as $n > \sqrt{n}\log n$ and]

5) We can make an algorithm similar to Quicksort but instead of random selection, we select the median as Pivot.

Given n elements:

- find median in $O(n)$ time.
- Generate two sets S, L, where S has elements smaller than median and L has elements larger than median.

  This takes $(n-1)$ comparisions.

The desired sorted output is

$$\text{Sort}(A[n]) = \{\text{Sort}(S[n/2]), \text{median}, \text{Sort}(L[n/2])\}$$

(as S, L have $n/2$ elements after splitting)

Total Time for this algo $\Rightarrow$ $[T(n) = 2T(n/2) + (n-1)]$

$\Downarrow$ similar to

$$T(n) = aT(n/b) + f(n)$$

$\therefore$ By master method,

as $f(n) = \Theta(n) = \Theta(n^{\log_2 2})$

$\therefore T(n) = \Theta(n^{\log_2 2} \lg n)$

$$\boxed{T(n) = O(n \lg n)}$$