SORTING

BY : NISHANT GUPTA

QUICK SORT

- The running time of the quick sort will lie between O(n logn) to o(n square).
- O(n log2n) will the best case is , means every time the data is divided equally , both sided.
- It remains in nlogn complexity till there is constant relation like, 1/10 on left and 9/10 on right , then O(n log 10/9n).
- O(n square) when the inputted data is either already sorted in ascending or descending way, as in that form a left or right one will have n-1 elements at every instant.

SORTING

- There is algorithm that can sort n random numbers in less than O(nlogn) complexity .
- T(n)={ solving trival problem if n=1;
- {
  num_pieces*T(n/subsiquent_sub_prob_size_factor) + dividing factor +combining factor
- } for n>1;
- Through this the recurrance relation can be made, for any general prob.

MERGE SORT[nlogn/ out place sorting algo]

- Merge sort(A,p,q)
- {
- If(p<q)
- R=p+q/2
- Merge sort(A,p,r)
- Merge sort(A,r+1,q)
- merge
- }

MERGE Function

- Merging of two sorted sequences if p1=n/2, p2=n/2 than it will take O(n) time complexity,
- Else if we have two sorted sequences to merge than if let, p1= l, p2=m, than the worst case time isO(m+l) and the best case running time is O(min(l,m));
- T(n)={O(1) if n=1;
- {2*T(n/2) + O(n)[for merging] if n>1;
- On solving it comes out to be nlogn for all the case means worst, average and best.

MERGE Function

- Merging of two sorted sequences if p1=n/2, p2=n/2 than it will take O(n) time complexity,
- Else if we have two sorted sequences to merge than if let, p1= l, p2=m, than the worst case time isO(m+l) and the best case running time is O(min(l,m));
- T(n)={O(1) if n=1;
- {2*T(n/2) + O(n)[for merging] if n>1;
- On solving it comes out to be nlogn for all the case means worst, average and best.

RADIX EXCHANGE SORT

- Sort the array with respect to the leftmost bit.
- Now partition the array at the point of changing of the bit.
- RECURSION :

Recursively sort the top subarray, ignoring the leftmost bit.

Recursively sort the bottom subarray, ignoring the leftmost bit.

Time Computation

- O(bn)
- T(n)= T(i,b-1) + T(n-I,b-1) + ‘n’[for the sorting of the leftmost bit! But y is it taking only O(n) time,]
- Answer is through bucket sort.
- On solving the above relation we gets O(bn).
- Radix sort is gud when b<logn,, but else quick sort is better.

STRAIGHT RADIX SORTO(bn)

- Examines the bits from right to left
- For(k=0, to b-1)

Sort the array in a stable way, looking only at bit 'k'.

What is the meaning of stable way?

It means that the relative order should be the same as before.

BUCKET SORT O(n)

- This type of sort works when we have numbers of duplicate numbers, and their values are small, like 1,1,2,3,2,1,3,2.
- Than one bucket is assign to every distinct number. And the rest of the values are added at the end of that bucket again and again in a sort of linked list.
- Total time is O(m+n), where ‘m’ is the total numbers of bucket, or the distinct numbers.