# LAB-4

Problem 1:

Theorem: Expected No of trials for success

$\boxed{1/p}$ $P(2) = 1/10$ $\frac{1}{1/10} = \underline{10}$

Theorem: Expected No for k success

$k/p = \frac{3}{1/10} = \underline{30}$

Problem 2:

Is there a comparision-based algorithm which, when run on array containing 4 elements require 4 comparisions?

Answer:- the possible arrangment of n elements is n!. As we observe from the Decision Tree for mergesort on 4 elements, the possible ordering of 4 elements is $4! \to 24$

→ Altho it might be possible to find the leaf node with 4 comparisions that is not always the case, the total comparision that we require is the depth of the longest leaf and the no of edges needed to reach there

So, 4 comparisions is not enough. but it can definitly be done by 5 comparisions.

⇒ from the Mathematical observation

$L \le 2^h$

$Leaf = 4! = 24$

$h \ge \lceil \log L \rceil = \lceil \log 24 \rceil = \lceil 4.585 \rceil = 5$

$\underline{h \ge 5}$

∴ 5 comparision are needed.

problem 3: Goofy algorithm
Step 1: Check if arr storted, If so return
Step 2: Randomly arrange the elements of arr
Step 3: Repeat step 1 & 2 until return

A- will Goofy work?
it might work, if we are really lucky. But there is no guarentee that this will ever sort the given input list.

B- what is the Best case for Goofy Sort?
the best case is If we can sort the cards in the first try.

C- what is the running time in the best case
• the best case running time is O(1), which is constant time.

D- What is the worst case running time.
the worst case is that we keep sorting and sorting and elements are never in a sorted order. that is $\infty$ our trial goes as big as $\infty$ without finding the sorted sequence.

E- What is the average case running time

It's really hard to determine the average running time for such algorithms. But I think the averge running time will be still $\infty$
Because there is no guarantee when we can find it.

F- Is the algorithm Inversion Bound?
• No it is not inversion Bound, B/C for a sorting to be inversion it has to prove inversions in L don't happen in Lr But here we are randomly inverting.

problem 4 :

A = [5, 1, 4, 3, 6, 2, 7, 1, 3]

a. which x in A are good pivots?

Sol$^n$

$\frac{3}{4}(9) = 6$ elements

- pivot - 5  L = 1 4 3 2 1 3  G - 6 7

  L = $\frac{3}{4}n$ ⇒ 5 - Bad

- pivot - 1  E - 1 1  L - 0 element  G - 7 element

  ∴ Bad pivot

- pivot - 2  E - 2  L - 2 elements  G - 6 elements

  ∴ Bad pivot.

- pivot - 3  E - 2 elements  L - 3 elements  G - 4 elements

  ∴ good pivot

- pivot - 4  E - 1 element  L - 5 elements  G - 3 elements

  ∴ good pivot.

- pivot - 6  E - 1 element  L - 7 elements  G - 1 element

  ∴ Bad pivot

- pivot - 7  E - 1 element  L - 8 elements  G - 0 element

  ∴ Bad pivot

So, the only good pivots are 3,3 and 4.

b. is it true that at least half of the elements of A are good pivots?

No, they are not in this case Because

$n/2 = 4$ or 5

But we got 3 good pivots

problem 5

Devise sidewaySorting that put elements of length-n integer array arranged

pos 0 -> small
pos 1 -> large
pos 2 -> 2nd small
pos 3 -> 2nd large

Algorithm sideWaySort (A)
Input: Ordered Array A From MergeSort algorithm
Output: sideWay Sorted output.

```
~~newArray~~
i ← 0
j ← A.length - 1
For k ← 0 to A.length-2 do
    if (i != j) then
    newArray [k] ← A[i]
    newArray [k+1] ← A[j]
        i ← i+1
        j ← j-1
      { k ← k+2 }
```

A. what is the asymptotic running time?

The asymtotic running time would be
O(n logn) from the MergeSort part and additional O(n) work to side sort

$$\Rightarrow O(n\log n) + n) = O(n\log n)$$

B. prove that it is impossible to obtain an algorithm to do sideway sorting of an intege array that runs asymptotically faster than the algorithm you created in part A.

proof: Any sorting algorithm that involves comparision and inversion of elements have ~~this property stated by the~~ the fastest running time of $O(n\log$

this is our lower Bound. we can't do better than this for sorting.