Home-Work 1 :

Author: Priyanka Dwivedi(pd6741@g.rit.edu)
Srikant Lakshminarayan(sxl8819@g.rit.edu)

1.)

Descending Order :

$1/N$

$1/1000$,10^{100} – same position

log log n

$\sqrt{\log n}$

log n , $\log_{10} n$, $\log n^2$ – same position

$\log^2 n$

$\sqrt{n}$

$n^{2/3}$

$\log 2^n$, $2^{\log n}$, n , n+log n – same position

n.log n

$n^{3/2}$

n^2 , $n^2/10^{20} + n$ – same position

2^n , 2^{n+1}

$n.2^n$

3^n

2^{2n} ,4^n – same position

n!

n+1 !

n^n

2)

a) If $f(n) = \Theta(g(n))$ then $2^{f(n)} = \Theta(2^{g(n)})$.

False.

For example: $f(n)=\log n$

$g(n)=\log n^2$

$2^{f(n)}= 2^{\log n} = n$

$2^{g(n)}= 2^{2\log n} = n^2$

b) If $f(n) = O(g(n))$ then $f(n)^2= O(g(n)^2)$.

True

We know that $f(n)<=c.g(n)$

Squaring ,

$f(n)^2<=(cg(n))^2$

Eg:- $f(n)=n$
$g(n)=n^2$

$f(n)^{2=}\ n^2$
$g(n)^2=n^4$

Therefore, $f(n)^2= O(g(n)^2)$.

c) If $f(n) = \Theta(g(n))$ then $1 /f(n) = \Theta(1/ g(n))$.

True
We know that If $f(n) = \Theta(g(n))$, then $f(n)=O(g(n))$ and $g(n)=O(f(n))$

If $f(n)=O(g(n))$ then $f(n) <= c.\ g(n)$

$1/g(n) <= c.1/f(n)$
$1/g(n) = O(1/f(n))$

Similarly, if $g(n)=O(f(n))$ then

$g(n) <=c.f(n))$
$1/f(n) <= c.1/g(n)$
$1/f(n) = O(1/g(n))$

Therefore $1 /f(n) = \Theta(1/ g(n))$.

d) $f(n) = \Theta(f(n - 1))$.

False

Eg:-
If $f(n)=(n+1)!$
Then $f(n-1)=n!$

Here is not $f(n)=O\ f(n-1)$

3.Planters
This algorithm follows a process where it finds out if each plant will get a bigger planter. First both the number of Existing plants and their sizes are taken in an array and merge sort is applied on both the arrays. After both the arrays are sorted then the common sizes and the largest size is taken in a temporary array and this array is compared with the array which has new sizes i.e. m new planters. this comparison is done in O(n). The algorithm

after sorting first, checks for different sizes in the current sizes array. As array is sorted, we can say that if element n-1 < n, then that n-1 element can get that n planter of bigger size. if there are same plans with same sizes, then it will transfer those plants into a temporary array. In this way, all those plants who dint get new plants now will be checked with the sizes present in the new array and if it is lesser then it will get a new planter else not. Also, the last element is the largest, so it will not have any new plater of a bigger size in the same array. so, the plants with same size and the largest plans are checked with the array of new m planters. now if the number of new planters and the number of plants remaining to get a new planter are same or the number of new planters is more than this will work fine. but when the number of existing plants left are more than the number of new plants, it will not even check the sizes, as all the old ones will not get a new planter. So, this algorithm will work for any kind of input, i.e. with different sizes as well as similar or same sizes.
Complexity : O(nlogn)

4. Stable Matching Algorithm:

In this algorithm, we pair people according to their preferences. For ex: Professor pairing with student or Men pairing with women. A Match is said to be unstable if there is other person who want to get paired up so for that preference is checked between current partner and new partner if new partner has higher preference then new partner will become his new pair and old partner would be set free.
The correctness of algorithm can be argued as while pairing students according to professor's preference we can start from any professor and end up with same output of pairs being formed. For ex: while pairing students according with professor's preference we can start pairing from any professor say 2 or 4 but we will end up pairing same irrespective who is paired first. In the algorithm implemented we have paired the professor and student according to professor's preference and vice versa. Then we compare how many same pairs are formed by pairing according to professor's preference and student's preference.

Complexity:
The running time complexity of Algorithm:
To get I/p from user it takes O(n) time.
To assign free professor it takes O(n) time.
For pairing professor, it takes O(n^2) time.

5.
b.

a. Random Floating point numbers with a Uniform Distribution

Note: All timing are in Seconds

Input size(N)	**Insertion Sort**	**Merge Sort**	**Bucket Sort**
100	0.0010461807250976562	0.0005848407745361328	0.00016188621520996094
1000	0.32120299339294434	0.007772922515869141	0.0014200210571289062
10000	37.812744140625	0.1939249038696289	0.04441499710083008
100000	Longer than 3 minutes	4.170264005661011	294.1784780025482
1000000	Longer than 3 minutes	28.920287099075317	Longer than 3 minutes

b. Random Floating point numbers with a Gaussian Distribution

Input size(N)	**Insertion Sort**	**Merge Sort**	**Bucket Sort**
100	0.0013821125030517578	0.0005869865417480469	0.00015807151794433594
1000	0.19990968704223633	0.0360109806060791	0.0015380382537841797
10000	29.692598819732666	0.21071195602416992	0.021589040756225586
100000	Longer than 3 minutes	2.4354097843170166	1503.8844690322876
1000000	Longer than 3 minutes	25.01664710044861	Longer than 3 minutes

Running time for Insertion Sort: O(n^2)
Running time for Merge Sort: O(n.logn)
Running time for Bucket Sort: O(n^2)