Ram011

Task 1. **Ps. Teaching assistant said I didn't have to worry that much about flooring and ceiling**

Function	**Exact**	**~**	**O()**
A	2n+1	~2n	O(n)
B	2 * log (n +1) + 1	~2 (log(n) +1)	O(n log n)
C	n+(n(n+1)/2)+2	~1/2(n^2+n)	O(n^2)
D	1	~1	O(1)
E	Ceil(1/3 * n)	~1/3n	O(n)
F	N*1/sum of k! and k = 1 through n	~	O()
G	N+1	~n	O(n)
H	(Log (n)+1) +1	~log (n)	O(log n)
I	N*(log n+1) when n=2^x	~n log(n)	O(n log n)
J	2*ceil(sqrt(n))	~2 sqrt(n)	O(n)
K	Log n +1 when n = 2^x	~log n	O(log n)
L	2n+2	~2n	O(n)
M	2n when n = 2^x	~2n	O(n)
N	Log n when n = 2^x	~log n	O(log n)
O	3^(n+1) / 2	~3^(n+1) / 2	O(n^4)
P	$N*\log_3(n) + n$	~ $N*\log_3(n)$	O(n log (n))
Q	(2*Log (n)+1) + n + 1 when n = 2^x	~n+2*Log (n+1)	O(n log n)
R	N^3…	~n^3+	O(n^3)
S	Log (n+1) (log)	~n	O(n)
T	2n+1+log(n) when n = 2^x	~2n	O(n)
U	Sqrt(n) + n + log n	~n+sqrt(n) + log n	O(n)
V	N*1/2n = ½ n^2	~1/2n^2	O(n^2)

Task 2.

a)

1) 0.5 * (500/100) = 2.5ms

2) 0.5 * (500/100) * log(500/100) = ~5.8ms

3) 0.5 * (500/100)^2 = 12.5ms

4) 0.5 * (500/100)^3 = 62.5ms

b)

1) 100/x = 0.5 / 60 000 x = 12 000 000 elements.

2) 100 log(100) / x log(x) = 0.5 / 60 000 x = 3 656 807 elements.

3) 100^2 / x^2 = 0.5 / 60 000 x = 34 641 elements.

4) 100^3 / x^3 = 0.5 / 60 000 x = 4 932 elements.

Ram011

Task 3.

b)

Path compression makes deunion hard because when using path compression, every node in a component points to the top root of that component. When deunifying you then have to change the root of all nodes in the whole (deunified)component instead of just changing the root of the top node.

Task 4.

a)

The amount of comparisons needed when sorting this way is n(n-1)/2.

c)

This method also has n(n-1)/2 comparisons.

Task 5.

e)

I started by testing values for A between 0-100 with increments of 2. This yielded unexpected results as it seems that the best run-time occurs when A is in the 30-90 range(tested this 10 times for each A and ran the program 10 times more after having weird results). These results are weird as I expected that when A is in the range of 10-30 to yield the best results.

Next I changed the increment from 2 to 1, but I still had the best run-time in 30-90 range. Here are some of the actual runtimes:

Value for A	Run-time
20	0.1233s
40	0.1167s
60	0.1165s
80	0.1169s

I settled on A = 50.

Next I tested values for B. After decreasing the increment value and range I eventually found that a B value around 1000 gave the best run-time.

Value for B when A=50	Run-time
500	0.1140s
1000	0.1089s
1500	0.1101s
2000	0.1113s

Next I tested values for C. The run-time doesn't seem to change all that much when c > 10 000 as it was fluctuating between 0.107s-0.110s all the way from C=10 000 to C=500 000. Here are some values:

Ram011

Value for C when A=50 and B=1000	Run-time
5000	0.1162s
50 000	0.1083s
100 000	0.1082s
250 000	0.1082s

To conclude:

A=50, B=1 000, C=200 000 worked the best for me, although results may vary from time to time and from computer to computer.