Yonatan Rubin 648831051
Eitan Brown 346816549

Q1
PART A AND B

```
For j=0 to n-1
{
  Int maxValue=value first slot of the array  (array[0])
  Int maxSlot=0
 //Both of these lines effectively run n times

  For i=0 to n-1-j
  {
    Compare array[i] with current max value
    If value at array[i] is greater than maxValue, maxValue is replaced with array[i] and maxSlot=i
    Else continue
    //all commands inside this loop run n^2 times
  }
  //above loop runs n times
    Swap array[n-j] with array[maxSlot]
    //runs n times

}
```

TEST CASE
//Array 1 8 2 7 5 OG
//maxval 1 max slot0
//maxval 8 maxslot1
//swap array [n-j-1] with arraySlot j=0, n=5

//1st run: j=0, n=5 last slot -->18275
//2nd run: j=1, n=5 second last --> 15278
//3rd run: j=2, n=5 third last --> 15278
//4rth run: j=3, n=5 fourth last --> 12578
//5th run: j=4, n=5, fifth last --> 12578
PART C
N^2

Q2
Arrange the follow-ing functions in ascending asymptotic order.
Please note if and where there is any asymptotic equivalence (that is θ).
Prove 2 of the cases.
5, lg2n, nlgn, 2n , (n+1)!, lglg n, n, n 2 , 5n , n!
Answer:
5 (fastest)

lg(lg(n))
lg2(n)
n
n lg(n)
n^2
2^n
5^n
c1<c2, c1n<c2n
Assuming c1 and c2 are positive and n>=1 → 8 will always be before 7
n!
(n+1)! (slowest)
9 and 10 are equal (equivalence aka
theta (n+1)!∈Θ(n!))

Q3
Question 3 For each pair of functions, determine whether
1. f(n) = O(g(n)),
2. f(n) = Omega(g(n))
3. f(n) = Θ(g(n)).
Prove your answers.
a. f(n) = nlg n +5n   g(n) = 5nlog3n5
b. f(n) = 2n   g(n) = 22n
Answer for A:
(1) True BIG O
n*Lg(n)+5n<=c*5n*log3(n5)

n*Lg(n)+5n/5n*log3(n5) = n*Lg(n)/5n*log3(n5) + 5n/5n*log3(n5)

Lg(n)/5*log3(n5)+ 1/log3(n5)<=c

If n=3 this simplifies ‘nicely’
Lg(3)/25+1/5 <= c
Positive number+positive number=positive number, therefore c is valid for n=3
The limit of the left hand side is about 0, making 0.2 a valid choice for c while n0=3

(2) OMEGA - False
Lg(n)/5*log3(n5)+1/log3(n5)≥c
As we showed last time the limit of the left hand side being 0, meaning c cannot exists as it must be positive (c>0)
This is invalid
(3) Theta (both one and two must be true) so it is also false
We showed in #1 that f(n)=O(g(n)), and in #2 that f(n)!=omega(g(n)), therefore f(n) cannot = θ(g(n))

Answer B:
(1) f(n) = O(g(n)) - True
2^n <= c*2^2n __> 2^n/2^2n <= c >__> 2^n-2n <=c >__>   2^-n <= c
As n→∞ 2-n=0 making this true when n=2 (random number) and c=¼
1/4 <= 1/4 GOOD

(2) f(n) = omega(g(n))
 2^-n >= c
Same as last time, left side goes to 0, c must also be 0 for this to be true, since c cannot be 0, the equation f(n) = omega(g(n)) is wrong

(3) Theta: To be true Omega and Big O must be true
We showed in #1 that f(n)=O(g(n)), and in #2 that f(n)!=omega(g(n)), therefore f(n) cannot = θ(g(n))

Q4 - Run time complexity of 4 examples:
1) f(n)
n^2 / 2

2) g(n,A)
4n^2+n

3) h(n)
n/3

4) q(n)
n(log4(n)+n^2)

---

q5
A
$f(n)=n^{1/3}$
$g(n)=n^{1/4}$
$c_1 n^{\frac{1}{4}} \leq n^{\frac{1}{3}} \leq c_2 n^{\frac{1}{4}} \qquad c_1 \leq n^{\frac{1}{3}-\frac{1}{4}} \leq c_2 \qquad c_1 \leq n^{\frac{7}{12}} \leq c_2$
c_1 can be anything since $n^{7/12}$ has a limit of infinity which means c_2 cannot exist as it must be constant!=∞
Disproved
(logically, since they have different order of magnitude, you cannot say one can be bounded by another using constant since one will be infinity greater)

B

n! = theta(nlg n)

$c_1 n\ *\ lg(n)\ \le\ n! \le c_2 n\ *\ lg(n) \rightarrow$

$c_1 \le \frac{n!}{n*lg(n)} \le c_2 \qquad c_1 \le \frac{(n-1)!}{lg(n)} \le c_2$

lg(n) is orders of magnitude smaller than (n-1)! (which is essentially the same as n!)
Same logic as before, n! is orders of magnitude bigger than any log function and cannot be bounded by a constant multiple of a less magnitude function

C

There is a positive function f such that f(n)=omega(log(n)) and $(f(n))^2$=O(f(n))
Logn is lower bound f(n)
f(n) is upper bound to (f(n))^2
This is not possible because $(f(n))^2$=O(f(n)) implies that f(n) is bounded by a constant from the formula $(f(n))^2$<=c*f(n)   f(n)<=c
since log(n) eventually reaches infinity meaning it is not bounded by a constant, a function that is bounded by a constant can't possibly be an upper bound of an infinite function

D

f(n)=log(n)
$f(n^2)=\log(n^2)=2\log(n)$
$c_1\log(n)<=2\log(n)<=c_2\log(n)$
$c_1<=2<=c_2$
c1=1 (why not) c2>=2
Therefore f(n)=log(n)

Q6

What is the run time complexity of func1 in terms of theta(n), Explain.

func1($\boldsymbol{n}$)

  i = 1

  j = 1

  while i<= n!

    func2(j)

    j = j + 1

    i = 2 $*$ i

func2($\boldsymbol{n}$) //this runs an n! Amount of times

i = 1

While i<=n //this runs n/2 amount of times

  i = i + 2

Answer:

F1: $\theta(\log_2(n!))$

F2: $\theta(n/2)$

Final answer: $\theta(\frac{n}{2} log_2(n!))$
