Name: KRISHNA - GUPTA Student ID: 920429478

457
8
8
$0 + 4 = y$

INSTRUCTIONS

You will use your y variable which is calculated by taking the last digit of your student id **mod 4** then add 4 to that result. Everything in the test is self-explanatory if you are confused state your assumption and answer the question. We will grade with your assumption in mind. **We cannot answer any questions related to your interpretation.**

What is your \$y\$?__4________

ANALYSIS - PROBLEM 1: (40 POINTS)

(a) 20 points Find the asymptotic run-time of the code below. Show the run-time line by line like we did in class and fill out the below questions to help answer.

```
1.void foo(int A[]){
2.    let n=A.size(); — O(1)
2a.   i=n;  — O(1)
3.    while(i>1) {
4.                    for(j=1 to n^2){          ] O(n^2)    O(log_4 n)
5.                          print "hello";
6.                    }//endfor
7.         i=i/y;
8.       }//endwhile
10.}//end foo()
```

lines 3-8: _$n^2 \log_4 n$_ why? The first two lines are O(1), and the outer loop which is a while loop is $(\log_4 n)$ and the nested for loop is $O(n^2)$, so since they are nested so total runtime complexity will be $\boxed{O(n^2 \log n)}$ Ans

final answer: _n^2__$\log_4 n$

(b) 20 points What is the asymptotic run time of foo? Fill out below to help answer the question. Solve for the closed form via the **recurrence tree method**.

```
1.void foo(int A[]){
2.    let n=A.size();
3.    for( i =  1 to n) {
4.                   for(j=1 to sqrt(n)){
5.                          i++;
6.                         print "hello";
7.                   }//endfor
8.     }//endfor
9.
10.   for(i=1 to y) {
11.        foo(A[1..A.size()/y])
12.   }//endfor
13.}//end foo()
```

— $O(1)$

$\boxed{O(\sqrt{n})}$ $O(n)$.

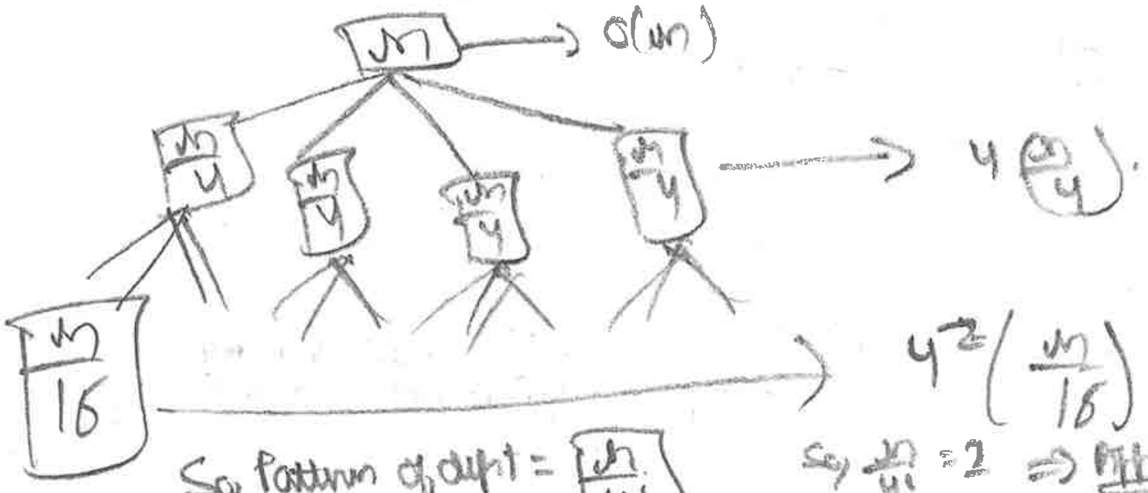
4 4

$4T\left(\frac{n}{4}\right)$.

lines 3-8: $O(n)$ why? The first loop is $O(n)$ as it goes (n) times, the nested loop goes $O(\sqrt{n})$ times so, ~~total~~ will be $\boxed{O(n)}$.

lines 10-12 $4T\left(\frac{n}{4}\right)$ why? In this $(y=4)$ So loop goes for 4 times and inside it is a recursive call of $\boxed{T\left(\frac{n}{4}\right)}$.

Recurrence equation for the work: $\boxed{4T\left(\frac{n}{4}\right)+O(n)}$
depth of recurrence tree: $\boxed{\frac{n}{4^i}}$ show work
work at each level: n show work
Asymptotic run-time: $O(n\log n)$ show work



So Pattern of depth = $\boxed{\frac{n}{4^i}}$ So $\frac{n}{4^i} = 1 \Rightarrow$ Apply log both sides. $\boxed{\log_4 n = i}$

Pattern for work at each step = $4^i \frac{n}{4^i} = \boxed{n}$

$$\sum_{i=0}^{\log_4 n} n = O(n\log n)$$

Analysis - Problem 2: (30 points)

Problem Description: Suppose you are choosing between the following 3 algorithms

Algorithm A Runs in T(n)=T(n-y)+1 time. Find the asymptotic run-time of algorithm A either by using the recurrence tree method or the formal master's theorem.

Algorithm B Runs in T(n)=yT(n/y)+6n time. Find the asymptotic run-time of algorithm B by any method.

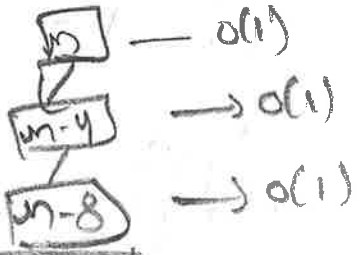
Algorithm C Runs in T(n)=(y+1)T(n/y)+6n time. Find the asymptotic run-time of algorithm C by any method.

Finally, order them from most asymptotically lowest to highest, and explain why.

$\boxed{\text{Given } y=4}$ $\boxed{\text{Algo-A}}$ $T(n) = T(n-4)+1$

Pattern dft = $\boxed{n-4i=1}$

$\boxed{\frac{n-1}{4} = i}$



$\boxed{\text{Pattern of work done} = 1}$

$$\sum_{i=0}^{\frac{n-1}{4}} 1 \Rightarrow \frac{n-1}{4} = \boxed{O(n)}$$

$\boxed{\text{Algo}=B}$ $T(n) = 4T\left(\frac{n}{4}\right)+6n$

$\boxed{\text{Applying masters theorem}}$

$A=4, B=4, d=1$

$n^{\log_4 4}$ (vs) n^1 $\Rightarrow$ So, its a trivial case

So, $n = \Theta(n) \Rightarrow \boxed{T(n) = O(n\log n)}$ yAns

$\boxed{\text{Algo}=C}$ $T(n) = (5)T\left(\frac{n}{4}\right)+6n$

Using master's theorem $\boxed{A=5, B=4, n^1}$

$n^{\log_4 5}$ (vs) n^1

$\boxed{n^{1.16} > n^1}$

$\boxed{\text{So, applying case-I}}$ $n = O(n^{\log_4 5 - \varepsilon})$.

$n = O(n^{1.16-\varepsilon})$. $\Rightarrow$ $\boxed{\text{Apply limit lemma}}$

$$\lim_{n\to\infty} \frac{n}{n^{1.16-\varepsilon}} = n^{1-1.16+\varepsilon} = n^{-0.16+\varepsilon}$$

$$\lim_{n\to\infty} \frac{1}{(n)^{0.16-\varepsilon}} = \boxed{0}$$

So, $\boxed{T(n) = O(n^{\log_4 5})}$ $\boxed{\varepsilon = 0.15}$

$\boxed{\text{Order}}$ $\Rightarrow$ $\boxed{O(n) < O(n\log n) < O(n^{\log_4 5})}$ = Ans

Algorithms - Problem 3: (10 points)

What is your \$y\$?___4______

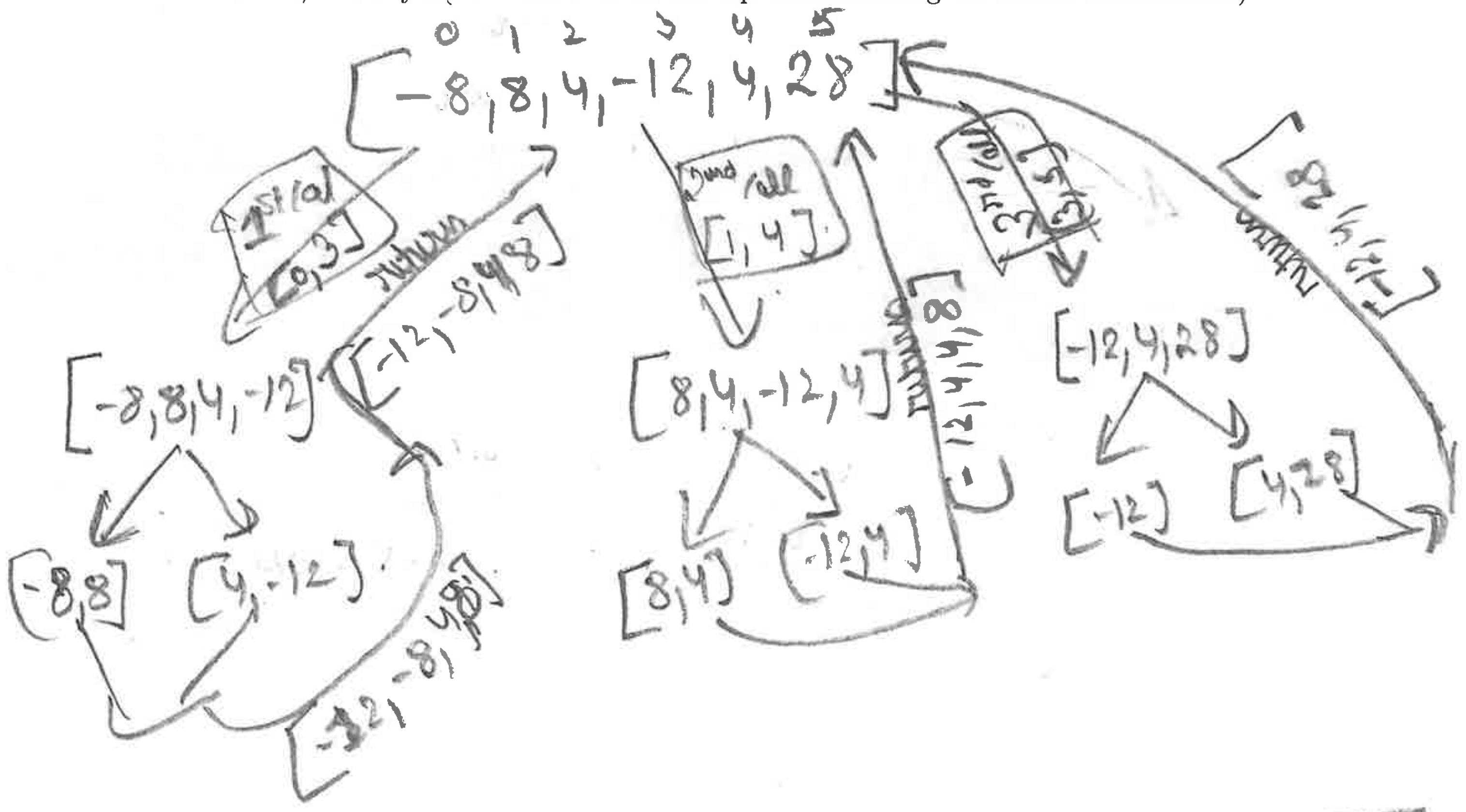
Given

A=[-2y,2y,y,-3y,y,7y]

For your y what integers does A contain? Use your A array to answer the question below

A=[-8, 8, 4, -12, 4, 28]

Show the work done by the algorithm at top-level recursion of mergeSort algorithm on array A. (In other words, show the input to the recursive calls at the top level, what is returned, and any other work done at the top level including the final return answer.)



So, final return after merge sort will [-12, -8, 4, 4, 8, 28] Ans

Design - Problem 4: (40 points)

a.)Design an interactive (non-recursive) most efficient algorithm that finds the largest and smallest number in an array of ints size n. (Analyze the runtime)

b.) Now Create a divide-and-conquer algorithm that finds the smallest and largest number in an array of ints with y recursive calls. So given A=[4,3,5] the algorithm would return 3 and 5.

b.) Analyze the run-time of your algorithm using recurrence trees assuming the base case is input of y or less.

c.) Confirm your result via substitution.

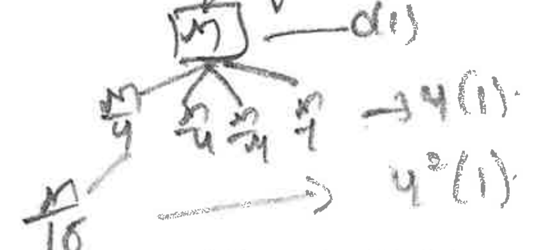
(a)
```
def smallest_~~and~~_largest(small=float('inf')), large=float('-inf'), arr[]):

    for a in arr:              ]
        if a > largest:        ] O(n)
            Largest = a        ]

    for a in arr:              ]
        if a < smallest:       ] O(n)
            smallest = a       ]
    return(smallest, largest) — O(1)
```
$\boxed{\text{Total runtime} = O(n)} = \text{Ans}$

(b) $\boxed{y=4}$, A: [4, 3, 5].
```
def small_largest(arr):
    if A.size == 1
    return A[0]
    if A.size = 2
        return small_largest(A[0], A[1])
    if A.size = 3
        return small_largest(A[0], A[1], A[2])
    else:
        a = small_largest(arr[1 —— n/4]
        b = small_largest(arr[n/4 + —— n/2]
        c = small_largest[arr[n/2 + n —— 3n/2]
        d = small_largest(arr[3n/2 —— n]
    return min(a, b, c, d), max(a, b, c, d)
```

(b) Runtime of algorithm $T(n) = 4T\left(\frac{n}{4}\right) + O(1)$



Pattern dept = $\frac{n}{4^i} \Rightarrow \frac{n}{4^i} = 1$
$\boxed{\log_4 n = i}$

Pattern Work done at each step $= 4^i$

$$\sum_{i=0}^{\log_4 n} 4^i \Rightarrow \frac{4^{\log_4 n + 1} - 1}{3} \Rightarrow \frac{n-1}{3} = \boxed{O(n)} = \text{Ans}$$

(c) Step 1: $T(n) \le ck$ | guess: $O(n)$

Step 2: $\boxed{n \le k-1}$ ~~held~~ $n = k$

$4T\left(\frac{n}{4}\right) + O(1) \le ck$

$4\left(c\frac{n}{4} - d\right) + a \le cn - d$

$c\cancel{n} - 4d + a \le \cancel{cn} - d$

$a \le 3d$

$\boxed{\frac{a}{3} \le d}$, $\boxed{c = 1}$ Ans

Design - Problem 5: (15 points)

You have an array of both negative and positive integers and your goal is to find a subset of ints such that the product of the subset is max. Please provide a brute-force algorithm and analyze it in terms of n- the input size.

```
def subset (arr):
  subset = [ ] — O(1)
    for i in range(len(arr)): —> O(n)
      new sub = [ ] — O(1)
      for sub in subset: ——> O(n)
        new sub = sub + [i] —> O(1)
      new sub.append(new sub) —> O(1)
    subset.append(new sub) —> O(1)
              product = 1
              for a in subset: ——> O(n)
                product = product * a[k] —> O(1)

      return max(product) —> O(1)
```

$\boxed{\text{So, total runtime} = O(n^2)}$ = Ans