Submit other program (python) Man Yeung (Sam)

. When array is sorted, the runtimes will be faster because it will just go one direction until the end by half of the time. No matter what approach.

Global maximum will go the end. $\Theta(n^2)$

Linear seach will go the end. $\Theta(n^2)$

Divide and conquer will go one way to the end. half by half. $\Theta(\log n)$.

1c. The impact is that it is looking for true peak.

It is not guaranteed to find a peak.

Because the array may have all elements a

equal to each other or one side is alway

equal, then we cannot find a peak

this situation.

Man Yeung (Sam) 1/23/2020

$T(n) = aT(n-b) + f(n)$, $n>1$

$T(n) = \begin{cases} O(n^k), & \text{if } a<1 \\ O(n^{k+1}), & \text{if } a=1 \end{cases}$

$O\left(n^{k a^{\frac{n}{b}}}\right)$, if $a>1$

$a=1, b=2, k=1$

$T(n) = O(n^2)$

1. Implement the three different approaches we discussed in the class to find "a peak" of a given 1-dimensional array of integers, using your favorite programming language. An element is said to be a peak if and only if it is greater than or equal to its left and right neighbors.
   a. Global maximum, linear search, divide and conquer **(2 pts each = 6 pts)**
   b. Test and provide runtimes of each approach for an array of size 1 million random integers. How does the run time change when array is sorted? **(1 pt)**
   c. When definition of peak is changed to greater than its neighbors, instead of greater than and equal to, what impact does it have on the problem? Are you guaranteed to find a peak ? Provide proper reasoning to support your argument. **(3 pts)**
   d. How can you extend the peak finding problem to 2-dimensional array of integers, when a peak is said to be greater and or equal to all its four (left, right, top, bottom) neighbors. Provide an efficient implementation **(5 pts, extra credit)**

e. $T(n) = \Theta\left((\log n)^{\log \frac{3}{2}}\right)$

$S(m) = \Theta\left(m^{\log \frac{3}{2}}\right)$

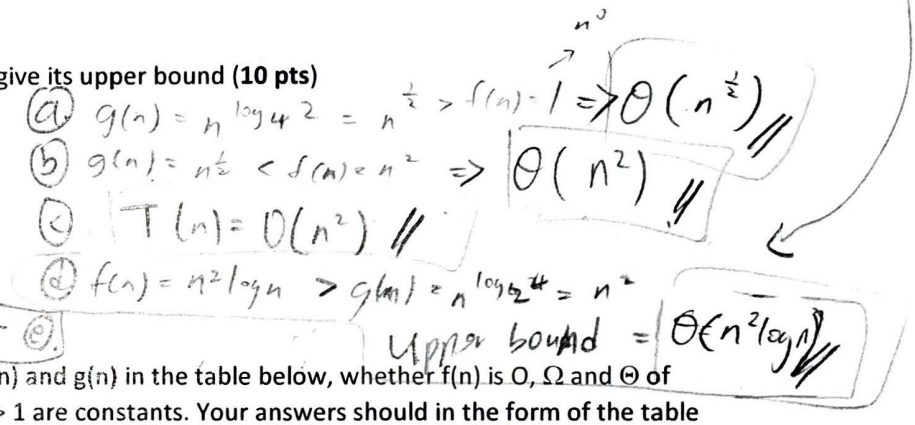
$m = \log n$

2. Solve the following recurrences and give its upper bound **(10 pts)**
   a. $T(n) = 2T(n/4) + 1$
   b. $T(n) = 2T(n/4) + n^2$
   c. $T(n) = T(n-2) + 2n$
   d. $T(n) = 4T(n/2) + n^2 \log n$
   e. $T(n) = 3T(\sqrt{n}) + \log n$

e. $T(2^m) = 3T(2^{\frac{m}{2}}) + m$

$S(m) = 3S(\frac{m}{2}) + m$

$g(n) = n^{1.58} >$ $f(n) = n$



3. Indicate, for each pair of functions f(n) and g(n) in the table below, whether f(n) is O, Ω and Θ of g(n). Assume that $k \geq 1$, $e > 0$, and $c > 1$ are constants. Your answers should in the form of the table with "yes" or "no" written in each box. **(3 pts)**

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$

$\log\left(\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n\right)$ ←

$\frac{1}{2}\log 2\pi + \frac{1}{2}\log n + n\log n - n\log e$

f(n)		g(n)	O (upper)	Ω (lower)	Θ (both)
$\log^k n$	<	n^e	Yes	No	No
n^k	<	c^n	Yes	No	No
2^n		$2^{n/2}$	No	Yes	No
log(n!)		$\log(n^n)$	Yes	Yes	Yes
$\sqrt{n}$		$n^{\sin(n)}$	No	No	No
$n^{\log c}$		$c^{\log n}$	Yes	Yes	Yes

4. Prove that running time of an algorithm is Θ(g(n)) if and only if its worst-case running time is O(g(n)) and its best-case running time is Ω(g(n)). **(2 pts)**

for $n \geq n_0$

Let $T(n)$ be the running time, if $T(n) = \Theta(g(n))$, then $0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n)$

As $0 \leq T(n) \leq c_2 g(n)$ for $n \geq n_0$, $T(n) = O(g(n))$, worst case

As $0 \leq c_1 g(n) \leq T(n)$ for $n \geq n_0$, $T(n) = \Omega(g(n))$, best case

↑ prove

Man Yeung (Sam)

$$O(1) < O((\log n)^2) < O(n \log n) < O(n\sqrt{n}) < O(n^2)$$

5. Arrange the following in order from smallest to largest. **(2 pts)**
   $O(n^2)$, $O(1)$, $O((\log n)^2)$, $O(n\sqrt{n})$, $O(n \log n)$

$O(1) < O((\log n)^2) < O(n \log n) < O(n\sqrt{n}) < O(n^2)$

6. What is the worst-case complexity for the code below? Show the recurrence relation **(3 pts)**

```
int someFunctionFoo(int n) {
    int count = 0;
    int cur = 1;
    while (cur < n) {
        count++;
        cur = cur * 2;
    }
    return cur;
}
```

$1 = \frac{n}{2^i} \Rightarrow n = 2^i \Rightarrow i = \log n$

1, 2, 4, 8, ... until cur > n, $\log_2 n$