Weekly Assessment 2

Instructions:

a. Write your answer in the corresponding code or text block.

b. To write equations, follow Latex formatting, such that writing the mathematical expression in between a pair of $ signs. Common expressions are:

- Superscript (exponent) x^2
- Subscript f_1
- Log $\log$
- Grouping literals n^{10}
- Greater than or equal to $\geq$
- Less than or equal to $\leq$

Example: $f(n) = 3n^{n+1} + 2n \log n + 10$

1. The number of operations executed by algorithms A and B is $8n \log n$ and $2n^2$, respectively. Determine n such that A is better than B for $n \geq n0$. [10 points]

###Answer:

let $8 \leq a$ where $a \in \mathbb{R}$

let $h(an) = 8n$, $j(n) = log(n)$, and $f_a(n) = h(an)g(n)$

recall "big O linear case" where $O(f(an))$ belongs to $O(f(n))$ therefore,

Function $h(an)$ belongs to $h(n)$

Next, $O(h(n)) = O(n)$ and $O(g(n)) = O(log(n))$

therefore, $O(f_a(n) = nlog(n)$

Moving on to B:

let $2 \leq b$ where $b \in \mathbb{R}$

let $b \leq k$ where $k \in \mathbb{R}$

let $f_b(n) = r(n)$ and $r(bn^b) = 2n^2$

Observe: function $O(r(bn^b))$ belongs to $O(r(n^k))$ therefore,

$O(f_b(n)) = O(n^k)$

Final Comparison:

$O(nlog(n)) < O(n^k)$

$nlog(n)$ has less value than n^k thus A takes less time,

therefore A is better than B for $n \geq n0$

2. Explain why the plot of the function n^c is a straight line with slope c on a log-log scale. [10 points]

Hint: Think of another way to write $\log n^c$

###Answer:

Start solution by re-writing to log scale,

Let $g(n) = n^c$

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

$\log(n^c) = c\log(n)$ from logarithmic property in math 21

Now solving for slope, observe the log-log slope equation:

$\log(f(n)) = m\log(n) + \log(b)$ where $m = slope$ and $b = intercept$

Therefore, if $\log(g(n)) = c\log(n)$, then the slope is $m = c$

because this equation is already written for log scale where m is a straight line, it can be concluded that $m = c$ is also the slope for a straight line

3. What is the sum of all the even numbers from 0 to 2n, for any positive integer n? [10 points]

Hint: Characterize this in terms of the sum of all integers from 1 to n.

Answer:

Starting from the hint,

Let summation from 1 to n be,

$S_a = 1 + 2 + ... + n$

$S_a = \Sigma(n)$

$S_a = n(n+1)/2$

Observe that: 2n contains only even numbers under it, therefore the sum of all even numbers is all values from (1 to n) multiplied by 2

$S_b = (2(1) + 2(2) + ... + 2n)$

$S_b = \Sigma(2n)$

$S_b = 2\Sigma(n)$ recall from Math 22 that constant can be moved outside of summation

$S_b = 2S_a$

$S_b = 2(n(n+1)/2)$

$S_b = n(n+1))$

Conclusion:

Summation of even numbers from 0 to 2n is $n(n+1))$

4. Order the following functions by asymptotic growth rate: [15 points]

- $4n \log n + 2n$
- 2^{10}
- $2^{\log n}$
- $3n + 100 \log n$
- $4n$
- 2^n
- $n^2 + 10n$
- n^3
- $n \log n$

###Answer:

- 2^{10}
- $4n$
- $3n + 100 \log n$
- $n \log n$
- $4n \log n + 2n$
- $n^2 + 10n$
- n^3
- $2^{\log n}$
- 2^n

5. Show that 2^{n+1} is $O(2^n)$. [10 points]

Hint: $2^{n+1} = 2 \cdot 2^n$

```
###Answer:

From hint,

$2^{n+1} = 2 \cdot 2^n$

From Dominating terms lecture,

let $2*2^n=a2^n$

let $0\le a \le d$ be true for all $0\le f(n) \le dg(n)$ where $g(n)=2^n$
```

```
Solving the inequality,

$a2^n \le d2^n$

$a \le d$

Finally, because $a$ is within range of $d$, the equation $2*2^n$ belongs to $O(g(n))$

Therefore,

$O(2*2^n) = O(2^n)$
```

6. Describe an efficient algorithm for finding the ten largest elements in a sequence of size n. What is the running time of your algorithm? [15 points]

###Answer: I will use a modified version of insertion sort. Enumerated below is the list of actions that will be done.

Assumptions:

a) Sequence is non repeating

b) Sequence data is increments by 1

c) Sequence has no gaps in between 1 to n] i.e. (1,4,3,5,2)

d)sequence has a minimum of 10 inputs always

1) first declare a list of from (n-9) to n

2) implement standard insertion sort such as from last lecture

3) compare sorted list to new_list[] at the end of every loop

4) since comparison is a O(1) action, the efficiency of the insertion sort will remain the same, at $O(n^2)$

7. Implement your algorithm for #6. [15 points]

In [ ]:
```python
# YOUR CODE HERE
#sample input
input=[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

def homework_insertionsort(to_sort2):
    #declare top 10 numbers
    n=len(to_sort2)
    target_list=[n-9, n-8, n-7, n-6, n-5, n-4, n-3, n-2, n-1, n]

    #start of standard insertion sort learned from lecture
    to_sort=to_sort2.copy()
    for j in range(len(to_sort)):
        item=to_sort[j]
        i = j-1
        while i>-1 and to_sort[i] > item:
            to_sort[i+1] = to_sort[i]
            i=i-1
        to_sort[i+1] = item

        #break condition for detecting the top 10 numbers in the list being sorted
        if(to_sort[-10:]==target_list):
            break

    return to_sort
```

8. Given the code below, determine the big-Oh running time of (a) example1 function, (b) example2 function, and (c) example3 function. [30 points]

In [ ]:
```
def example1(S):
  """Return the sum of the elements in sequence S."""
  n = len(S)
  total = 0
  for j in range(n):              # loop from 0 to n-1
    total += S[j]
  return total

def example2(S):
  """Return the sum of the elements with even index in sequence S."""
  n = len(S)
  total = 0
  for j in range(0, n, 2):        # note the increment of 2
    total += S[j]
  return total

def example3(S):
  """Return the sum of the prefix sums of sequence S."""
  n = len(S)
  total = 0
  for j in range(n):             # loop from 0 to n-1
    for k in range(1+j):         # loop from 0 to j
      total += S[k]
  return total
```

Answer:

Answer for a: $O(a(n)) = O(n)$

Answer for b:

let $b(n) = n/2 = bn$ due to increment of two

then with the inequalities,

$0 \leq b \leq d$ be true for all $0 \leq f(n) \leq dg(n)$ where $g(n) = n$

$bn \leq dn$

$b \leq n$

we can noq conclude,

then $O(b(n)) = O(bn) = O(n)$

Answer for c:

Observe: size j is actually equal to size n therefore,

$O(c(n)) = O(n * j) = O(n * n) = O(n^2)$

$O(n^2)$ is under $O(n^k)$ for all $0 \leq 2 \leq k$ for all $0 \leq f(n) \leq kg(n)$ where $g(n) = n^k$

Therefore final time is: $O(n^k)$