Sean Morrissey

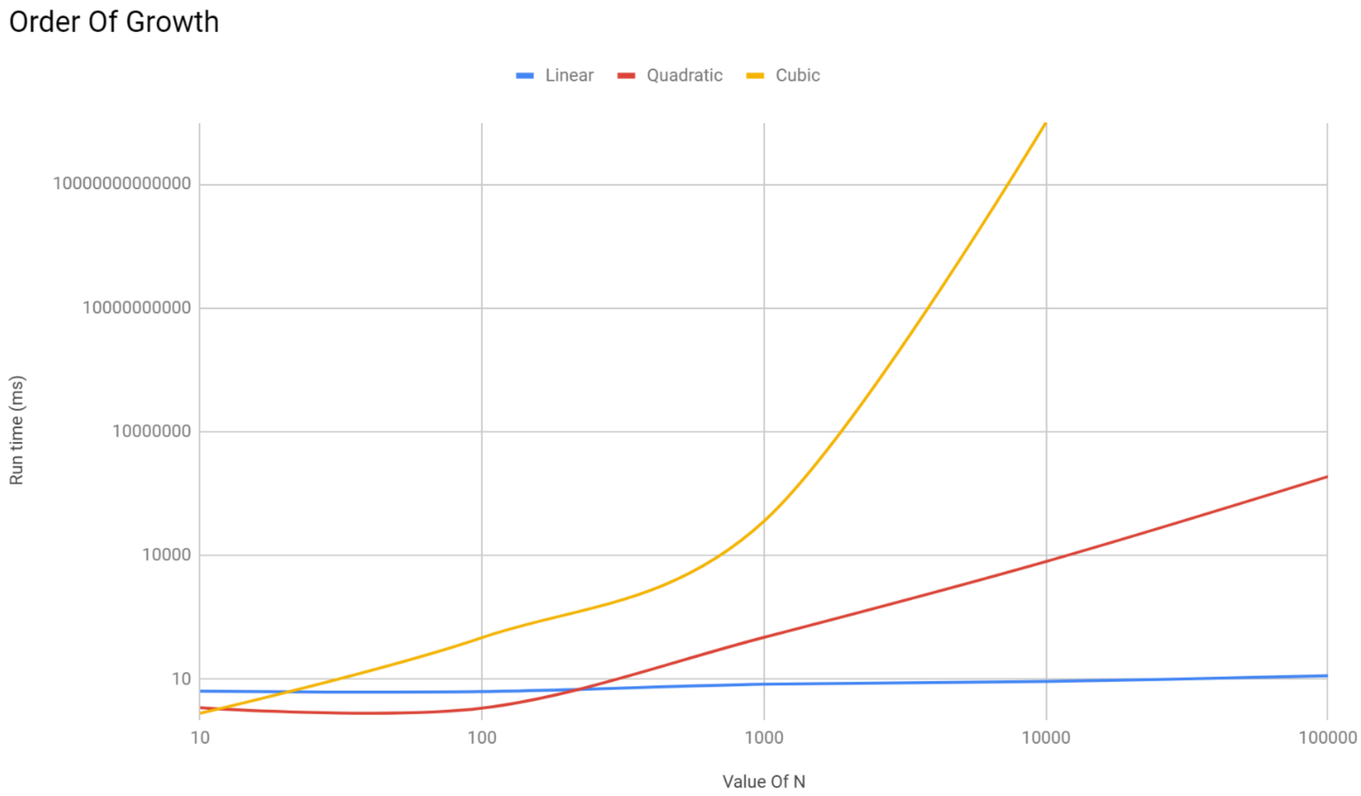
Professor Ramoza Ahsan

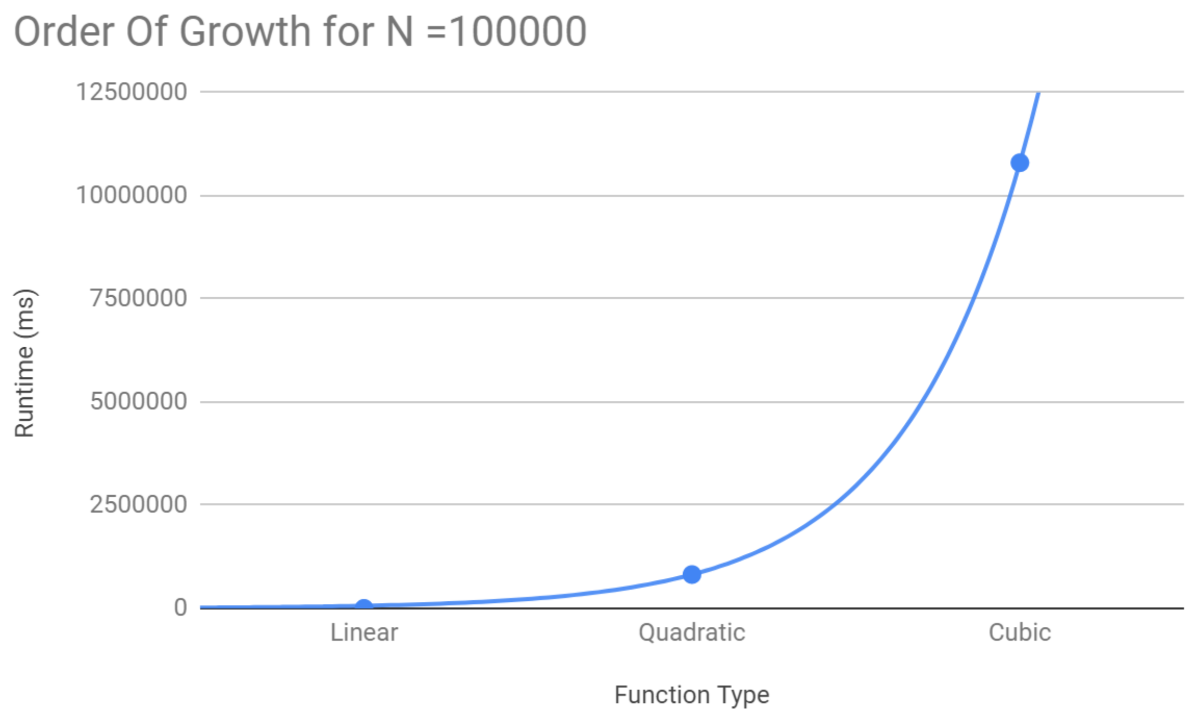
CS 2223 Algorithms

24 March 2019

Assignment 1 Report

Question 1.





Question 2.

```
For i = 1 to 4
    Split the quarters into groups of $\frac{81}{3^i}$ quarters
    If the first group and Second group of quarters weigh the same
        Then get rid of the quarters on the balance
    Else if the first group weighs less than the second group
        Then get rid of the second group and third group
    Else get rid of the first group and third group of quarters
```

Question 3.

The order of the given functions by increasing order of growth is as follows:

1. $\frac{1}{n}$
2. $2log_2 n$
3. $12\sqrt{n}, 50n^{0.5} + 50$
4. $n^{0.53}$
5. $3^{22}n$
6. $nlog_2 n$

7. $n^{log_2 n}$
8. $n^2 - n, 50n^2 + 8n + 50$
9. $2n^3 + n$
10. 3^n

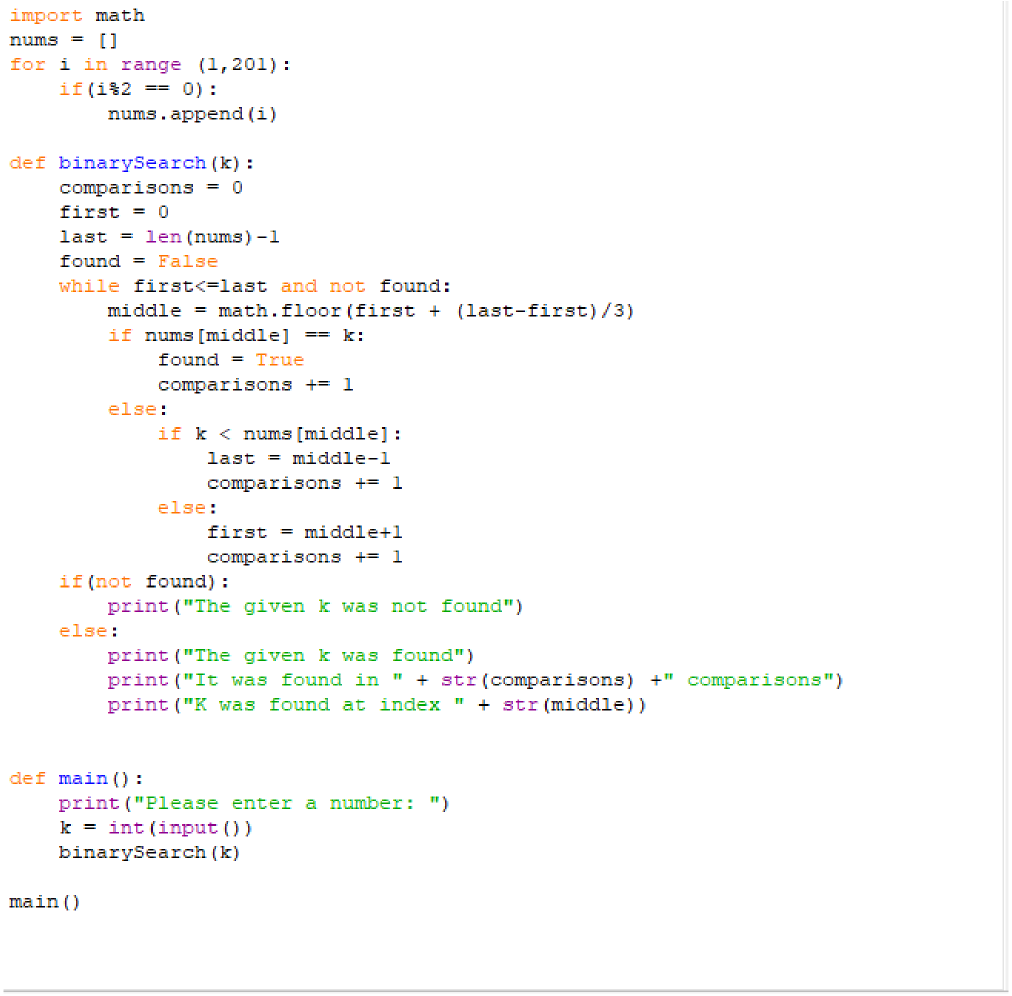
The following have the same Θ order for a runtime of $n^{0.5}$:

$12\sqrt{n}, 50n^{0.5} + 50$

The following have the same Θ order for a runtime of n^2 :

$n^2 - n, 50n^2 + 8n + 50$

Question 4.



```
import math
nums = []
for i in range (1,201):
    if(i%2 == 0):
        nums.append(i)

def binarySearch(k):
    comparisons = 0
    first = 0
    last = len(nums)-1
    found = False
    while first<=last and not found:
        middle = math.floor(first + (last-first)/3)
        if nums[middle] == k:
            found = True
            comparisons += 1
        else:
            if k < nums[middle]:
                last = middle-1
                comparisons += 1
            else:
                first = middle+1
                comparisons += 1
    if(not found):
        print("The given k was not found")
    else:
        print("The given k was found")
        print("It was found in " + str(comparisons) +" comparisons")
        print("K was found at index " + str(middle))


def main():
    print("Please enter a number: ")
    k = int(input())
    binarySearch(k)

main()
```

Analysis of the Algorithim:

Because we are working with a binary search, the worst case for the binary search algorithm is when no match for k is found in the array or the array is divided until one item, k, is found. In addition, we know that in a normal binary search tree that the big O run time is

O($log_2 n$) due to recursively dividing the list in half while performing constant operations for each iteration. But, in this case it is slightly different. On every iteration we calculate the "middle" element to be $n * (\frac{1}{3})$, n being the length of the array or subarray subtracted by one. Therefore, we can say with every iteration k we make $\frac{n}{3^k}$ comparisons. Since we are looking for one element, $\frac{n}{3^k} = 1$. When solving for k, we get that the number of iterations made is $log_3 n$. Since we are looking for what the worst case run time would be, it would mean that we made $log_3 n$ iterations or O ($log_3 n$) runtime.

Question 5.

```
CheckProduct (A,t)
    MergeSort (A)
    For i = 1 to n
        If (BinarySearch (A, t/A[i])) then
            Return True
        End If
    End For
    Return False
```

Analysis of Algorithm:

First, a call is made on Array A using Merge Sort, whose worst case is O ($nlogn$). Then a for loop is called from 1 to n, the size of the array, adding n to the runtime. In this for loop, there is an if statement which calls for a Binary Search, an algorithm with worst case runtime of O ($logn$). Therefore, if we add the run time of the for loop to the sort then the runtime is $nlogn$ + $nlogn$, which we can simplify to being 2nlogn. If we then drop the constants, the worst case runtime is O ($nlogn$).