Xueying Chen
OUID: 113366600
DSA4413 – Fall 2019
Homework #3

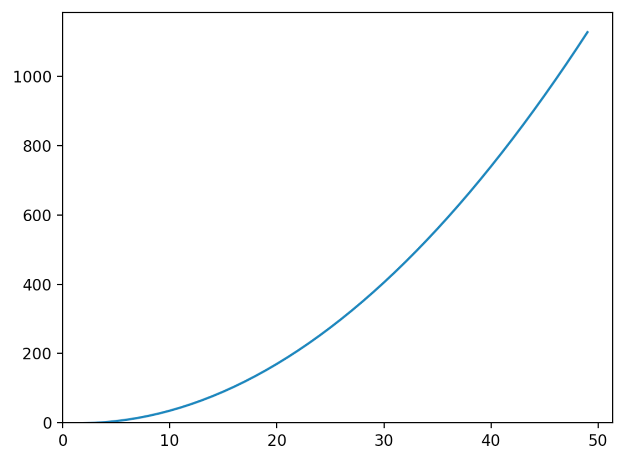
1a)

For selection, each iteration we have (n-1) comparison,

T(n) = (n-1) + (n-2) + (n-3) + …. + 3 + 2 + 1
T(n) = sum(i, for i=1 to (n-1))

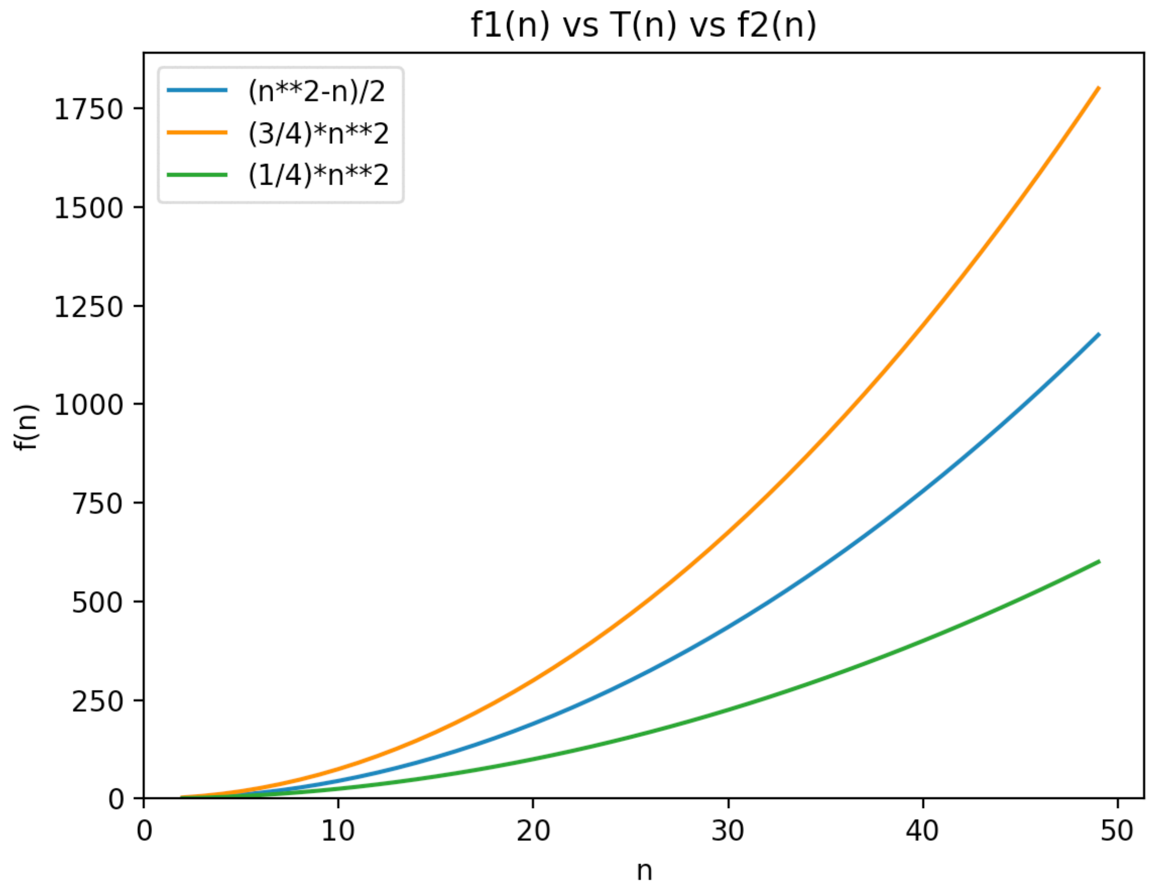
Form HW2:
sum(i, for i=1 to n) = n(n+1) /2
sum(i, for i=1 to (n-1)) = **n(n+1) /2 – n = ½(n^2 +n – 2n) = 1/2 (n^2 – n)**

lim (**1/2 (n^2 – n)/n^2**) n -> inf
= ½(inf^2 – inf) /inf^2 = ½



Plot T(n) = **1/2 (n^2 – n)**

1b)



F2(n) < T(n) < f1(n) for all n>=2

2)

asymptotically non-negative function: a function increase or decrease until it approaches some fixed value

Theta(g(x)) = {f(x): if there exist positive constants c1, c2, and x1 such that 0<=c1(g(x)) <= f(x) <= c2(g(x)) for all x>=x1}

g(x)= f1(x) + f2(x) + f3(x) + f4(x) + f5(x)

f(x) = max (f1(x),f2(x),f3(x), f4(x), f5(x))

prove:
c1(g(x)) <= f(x) <= c2(g(x)) for all x>=x1

divide by g(x)

c1 <= f(x)/g(x) <= c2

f(x)/g(x) <= c2 hold for c2 > 1 since f(x) < g(x)

c1 <= f(x)/g(x), asymptotically non-negative function will approach fixed value as x gets large, when x gets large x->inf, f(x)/g(x) -> 1, since f(x) is max (f1(x),f2(x),f3(x), f4(x), f5(x))

Therefore, we can find a positive number n such that
c1 = 1/n <= f(x)/g(x)

Therefore
c1(g(x)) <= f(x) <= c2(g(x)) for all x>=x1 for c1=1/n (n is large number). and for c2 > 1
and

Theta(f1(x) + f2(x) + f3(x) + f4(x) + f5(x)) = f(x) = max (f1(x),f2(x),f3(x), f4(x), f5(x))

3)

See Chen_Xueying_HW3_3.py for implementations,
to run the program, "python Chen_Xueying_HW3_3.py"

Pseudo Code using merge sort

```
input: array A
fount = found
// k is value to to find
int k

// mergeSortFind sorts array A using merge sort, try to find K which equals the summation of 2
integer in A.
// If k is found, print the value and exit
mergeSortFind(array A, k)
        if length of A <= 1 then
                return A
        let m = middle index of A
        let left = A[0 to m]
        let right = A[m to size of A]
        mergeSortFind(left, k);
        mergetSortFind(right, k);
        if not found:
                merge(left, right)
                value = findK(left, right, k)
                // if K is found
                if value != null:
```

```
                    found = true
                    print(value)

// find K which equals the summation of 2 integer in left and right array
// left and right are sorted array
findK(left, right, k) :
    low_index = first index of left
    high_index = last index of left

    while low_index < len of left and right_index > 0
        sum = left[low_index] + right[high_index]
        if sum == k
            return low_index, high_index
        if sum < k
            increase low_index by 1
        if sum > k
            increase high_index by 1
```

Analysis:

This is use mergesort to sort array A, and try to find K during the sorting process. We all know mergesort is O(nlogn). I add a function findK to find K in sorted sublists. findK is pretty much another merge operation, time complexity is n in the worst case.

Algorithm to findK in sorted array, we have low_index and high_index. We loop thru array, if the sum of the values at low_index and high_index equals to k, we are done. Otherwise, increase low_index if sum is less K and decrease high_index if sum is greater than K. Time complexity is n in the worst case.

Therefore mergeSortFind is O(nlog(n)

4)

a)

k=1, T(n) = T([n/2]) + 1

k=2, T([n/2]) = T([n/4]) + 1 → T(n) = T([n/4]) + 1 + 1 → T([n/4]) + 2

k=3, T([n/4]) = T([n/8]) + 1 → T(n) = T([n/8]) + 1 + 1 + 1→ T([n/8]) + 3

T(n) = T([n/2^k]) + k

when n = 2^k – 1

n + 1 = 2^k

k = log(n+1)

T(n) = T(2^k – 1 / 2^(k-1)) + k – 1

$T(n) = \log(n+1)$

b)
$k = 1, T(n) = T(\mathrm{sqrt}(n)) + 1$
$k = 2, T(\mathrm{sqrt}(n)) = T(\mathrm{sqrt}(\mathrm{sqrt}(n)) + 1 \rightarrow T(n) = T(\mathrm{sqrt}(\mathrm{sqrt}(n)) + 2$

$T(n) = T(\ 2\text{^}k\mathrm{root}(n)\ ) + k$

$2\text{^}k\mathrm{root}(n) = 2 \rightarrow n = 2\text{^}(2\text{^}k) \rightarrow \log(\log(n))$

$T(n) = 1 + \log(\log(n))$

c)
$k = 1, T(n) = 3T(n/2) + 8n,\ \ n = 2\text{^}k,\ \ T(1) = 1$

$k = 2, T(n/2) = 3T(n/4) + 8(n/2) \rightarrow T(n) = 3(3T(n/4) + 8(n/2)) + 8n = 3\text{^}2\ T(n/4) + 3*8(n/2) + 8n$

$k = 3, T(n/4) = 3T(n/8) + 8(n/4) \rightarrow T(n) = 3(3(3T(n/8) + 8(n/4)) + 8(n/2)) + 8n = 3\text{^}3(n/2\text{^}2) + 8n$
$3\text{^}2/2\text{^}2 + 8n\ 3/2 + 8n$

$T(n) = 3\text{^}k\ T(n/2k) + 8n * \mathrm{sum}\ (\ (3/2)\text{^}i \text{ for } i = 1 \text{ to } k\text{-}1)\ )$

$k = \log(n)$

$T(n) = 3\text{^}(\log(n)) + 8\ \log(n) + \mathrm{sum}\ (\ (3/2)\text{^}i \text{ for } i = 1 \text{ to } k\text{-}1)\ )$

5)

a)
5^(5^n) != O(5^n),

5^(5^n) <= C*5^n for all n1 > n

as n get large, 5^(5^n) is exponentially larger than 5^n, therefore there is NO Constant C such that 5^(5^n) < C*5^n

5^(n+1) = 5*5^(n) = O(5^n)
5*5^(n) <= C5^n, C >= 5

b)
log*(logn) is asymptotically larger.

when n -> large, log*(logn) = log*(n), if we take log(log*n) again, then

log(log*n) <<< (exponentially less than) log*(logn),

therefore log*(logn) is asymptotically larger.

6)

a)

Each sublist has length z, worst-time to sort a sublist using insertion sort is O(z^2). There are n/z sublist,

T(n) = z^2*(n/z) = nz = O(nz)

b)

Recall in mergesort, we recursively call on n/2 elements until number of element is 1. For merge insertion sort, we will recursively call on n/2 until number elements = n/z. n/z is the number of sublist. Subproblem size is log(n/z) instead of log(n). Therefore, the sublists can be merged in

O(nlog(n/z))

c)

O(nz + nlog(n/z)) < O(nlog(n)) → O(nz) < O(nlog(n)) since nlog(n/z) is less than nlog(n)

the largest value of z is log(n), if z is greater than log(n), then
O(nz) > O(nlog(n))

d)

We just have to find the value when insertion perform better than merge sort in average case. The number is around 40. z < 40

Bonus Question:

run “`python Chen_Xueying_HW3_6.py`”