Name Leonardo Roman
CS111 homework 6

1.Trace selection sort on the following array of letters (sort into alphabetical order):

0	1	2	3	4	5	6
A	X	B	T	S	Q	C

Iteration 1: i = 6
j=0 ; j<i ; j++

0	1	2	3	4	5	6
A	C	B	T	S	Q	X

Iteration 2: i = 5
j = 0; j < i; j++

0	1	2	3	4	5	6
A	C	B	Q	S	T	X

Iteration 3: i = 4
j = 0; j < i; j++

0	1	2	3	4	5	6
A	C	B	Q	S	T	X

Iteration 4: i = 3
j = 0; j < i; j++

0	1	2	3	4	5	6
A	C	B	Q	S	T	X

Iteration 5: i =2
j = 0; j < i; j++

0	1	2	3	4	5	6
A	B	C	Q	S	T	X

Iteration 6: i = 1
j = 0; j < i; j++

0	1	2	3	4	5	6
A	B	C	Q	S	T	X

For every iteration on the outer loop there are i comparisons in the j-loop and after j-loop is done comparing, the i-loop decrements by one. Hence the number of comparisons is the linear sum of the outer loop $\sum_{i=1}^{n-1} i$. Since the size of the array is 7 in this case, then the number of letter-to-letter comparison is 6+5+4+3+2+1 = 21.

2.Trace insertion sort on the following array of letters (sort into alphabetical order):

j i

0	1	2	3	4	5	6
A	X	B	T	S	Q	C

i=1 ; j = 0
key = X

j i

0	1	2	3	4	5	6
A	X	B	T	S	Q	C

i=2 ; j = 1
Key = B

j i

0	1	2	3	4	5	6
A	B	X	T	S	Q	C

i=3 ; j = 2
Key = T

j i

0	1	2	3	4	5	6
A	B	T	X	S	Q	C

i=4 ; j = 3
Key = S

			j	i		
0	1	2	3	4	5	6
A	B	S	T	X	Q	C

i=5 ; j = 4
Key = Q

				j	i	
0	1	2	3	4	5	6
A	B	Q	S	T	X	C

i=6 ; j = 5
Key = C

					j	i
0	1	2	3	4	5	6
A	B	C	Q	S	T	X

As we can see as i moves up j follows, however j will move down j times every iteration (j-- at j = i-1). As j decrements one it compares index value with key(a[j] > key as j--). In this case there are 17 comparison for sorting the array.

a. Determine if 2 arrays contain the none of the same elements elements (assume all elements are distinct)

1. Algorithm pseudocode:

1. for(i = 0; i < array1.length; i++)
2. for(j = 0; j < array2.length; j++)
3. if(array1[i] == array2[j]): there exist an element in array1 that is in array2 stop.
4. else: there is no element in array1 that is in array2.

2. The factor that would influence the running time of this algorithm is determined by the size of length of both arrays. This factor can be described as n-by-m assuming the two arrays are different in length size.

3. Number of operations in this algorithm is:

1. Initializing i.
2. Comparing i against array1 length.
3. Incrementing i.

4. Initializing j.
5. Comparing j against array2 length.
6. Incrementing j.
7. Comparing array1[i] against array2[j].
8. Printing message.

4. The number of operations can be expressed as a function of the two length of the arrays.
$f(n, m) = n \times m$

5. Best case inputs are two array in which the first element in array 1 equals the first element in array two. Worst case no elements in array 1 exist in array 2. On the other hand, best input case might be comparing two arrays with only one element. And worst case might be comparing two infinitely large arrays, assuming in both cases no elements in array1 is in array2.

6. Big O derivation:

i-loop **j-loop**

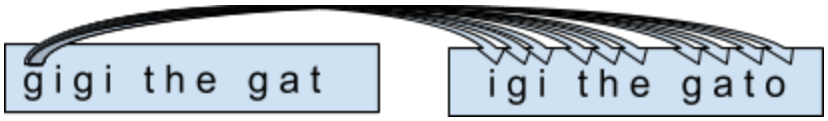
op	Time Cost	How many(success)		op	Time Cost	How many(success)
i = 0	1	1		j = 0	1	1
i<array length	1	best 1 worst n		i < array length	1	best 1 worst n
i++	1	best 0 worst n-1		j++	1	best 0 worst n-1
if statement	1	best 1 worst n		if statement	1	best 1 worst n

Best case $\rightarrow f(n, m) = nm = (1 + 1)(1 + 1) = 4$
Worst case $\rightarrow f(n, m) = nm = (3n - 1)(3n - 1) = 9n^2 - 6n + 1$
$f(n, m)$ *is* $O(n^2)$

b. Counting total number characters that have a duplicate within a string (i.e. "gigi the gato" would result in 7 (g x 3 + i x 2 + t x 2)

1. This algorithm uses two loops to compare the current character in a string with the next one.

1. for(i = 0; i < String length - 1; i++)
2. for(j = i+1; j < String length; j++)
3. if(String char at index i equals String char at index j AND it not space): count the letters



2. The factors that would influence the running time and which can be known before the algorithm or code is executed. For this algorithm, we can see that outer loop does n-1 iteration and so does the inner loop (n being the length of the string). Therefore, the factors that influence the time error for this algorithm can be represented as the input length of the string to be traverse. This factor can be represented as n-1.

3) Identify the operations that must be counted.

1. Initializing i.
2. Comparing i against String length -1.
3. Incrementing i.
4. Initializing j.
5. Comparing j against String length.
6. Incrementing j.
7. Comparing String.chatAt(i) against String.charAt(j).
8. Incrementing letter count.

4) Count operations by code Char = n(n+1)/2 duplicate counter = n loop = n

5) Determine best and worst case Worst case: if the array is all in descending order. This will require the code to go through every element to sort it. Best case: if the array is already in order.

6. Big O derivation:

i-loop **j-loop**

op	Time Cost	How many(success)		op	Time Cost	How many(success)
~~i = 0~~	~~1~~	~~1~~		~~j = 0~~	~~1~~	~~1~~

i<array length	1	best 1 worst n-1		i < array length	1	best 1 worst n
i++	1	best 0 worst n-2		j++	1	best 0 worst n-1
if statement	1	best 1 worst n		if statement	1	best 1 worst n
counter	1	best 1 worst n		counter	1	best 1 worst n

Best case $\rightarrow f(n,m) = nm = (1+1+1)(1+1+1) = 9$
Worst case $\rightarrow f(n,m) = nm = (4n-3)(4n-3) = 16n^2 - 24n + 9$
$f(n,m)$ *is* $O(n^2)$

c. Finding a row where every entry is 'a' in a 2-D array.

1. Algorithm pseudocode

1. for (i = 0; i<arr.length; i++)
2. for (j = 0; j<arr[i].length; j++)
3. if (arr[i][j] != 'a'): row i does not contain all a's break out the inner loop.
4. else: row i contains all a's stop.

2. The factor that would influence the running time of this algorithm is determined by the size of length of both rows and columns. This factor can be described as n-by-m assuming the 2-D array number of rows is different from the number of columns.

3. Number of operations in this algorithm is:

1. Initializing i.
2. Comparing i against array row length.
3. Incrementing i.
4. Initializing j.
5. Comparing j against array column length.
6. Incrementing j.
7. Comparing array[i][j] against key.
8. Printing message.

4. The number of operations can be expressed as a function of the two length of the arrays.
$f(n,m) = n \times m$

5. Best case inputs for failure is that the first element in every row is not a letter ‘a’. This way the, if first element is not an ‘a’ the program will break out of the inner loop and stop comparing, giving as a result a count number of number of rows in the 2-d array. Worst case input for failure would be, if all elements in every row is a letter ‘a’ except for the last element in every row. On the other hand, best case for success would be if the first row of the 2-D array contains all ‘a’. The worst case for success would be, if all elements in last row are all ‘a’.

6. Big O derivation:

i-loop **j-loop**

op	Time Cost	How many(success)		op	Time Cost	How many(success)
~~i = 0~~	~~1~~	~~1~~		~~j = 0~~	~~1~~	~~1~~
i<array length	1	best 1 worst n		i < array length	1	best 1 worst n
i++	1	best 0 worst n-1		j++	1	best 0 worst n-1
if statement	1	best 1 worst n		if statement	1	best 1 worst n

Best case $\rightarrow f(n,m) = nm = (1+1)(1+1) = 4$

Worst case $\rightarrow f(n,m) = nm = (3n-1)(3n-1) = 9n^2 - 6n + 1$

$f(n,m)$ *is* $O(n^2)$