Assignment: 04 Name: Methew Guda

Due Date: 16-11-2015 A#: A00381751

---

Q01.

Part (a) - It will take 1 comparison to find the target at first position, 2 comparisons to find the target at second and n comparisons to find the target at nth.

So total number of comparisons is 1 + 2 + 3 + … N = $\sum_{i=1}^{n} i$ = N(N+1)/2 and average number of comparisons is (N(N + 1)/2)/N = (N+1)/2. So average number of comparisons when there's a 75% chance that the target is in the list = 3(N + 1)/ (4*2)

Now, number of comparisons when target is not found is n and average number of comparisons is N/1 = N. So average number of comparisons when there's a 25% chance that the target is not in the list is N/4

So total average cost is 3(N + 1)/ (4*2) + N/4

Part (b) - When target is in the list, there's 75% chance that it's in first half of the list.

Number of comparisons when target is in first half of the list = $\sum_{i=1}^{n/2} i = \frac{n/2(n/2+1)}{2}$

Average number of comparisons in the first half of the list = $\frac{n/2(n/2+1)}{2*n/2} = \frac{(n/2+1)}{2} = \frac{n+2}{4}$

Number of comparisons when target is in the second half of the list = $\sum_{i=n/2+1}^{n} i = \frac{n(3n+2)}{8}$

Average number of comparisons in the first half of the list = $\frac{n(3n+2)}{8*n/2} = \frac{(3n+2)}{4}$

So total Average Cost = $\frac{3n+6}{16} + \frac{(3n+2)}{16} = \frac{(6n+8)}{16} = \frac{(3n+4)}{8}$

Q02.

## Part (a)

```
for i = 0 to n - 1 do
    for j = i + 1 to n do
        if A[i] > A[j]
            swap(A[i], A[j])
        end if
     end for
end for
```

Part (b) - Number of comparisons when i = 1 is n-1, number of comparisons when i = 2 is n-2 and finally number of comparisons when i = n-1 is 1.

So, total number of comparisons for a list of size n = $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$

Part (c) - There is 0 swaps when size of list is 1, 1 swaps when size of list is 2, 2 when size of list is 3 and (n - 1) when size of list is n.

Part (d) - Let there is d number of same element in the list size of n. Since there will be no swap in the smart swap algorithm if i = j. So the number of smart swap will be occur is (n - d).

Q03.

Recursive pseudocode algorithm for Selection Sort:

```
selectionSort (arrayList, startIndex)
        if (startIndex >= arrayList.length - 1)
        return;

        minIndex = startIndex;

        for (index = startIndex + 1; index < arrayList.length; index++) do
            if (arrayList[index] < arrayList[minIndex])
            minIndex = index;
        end for

        temp = arrayList[startIndex];
        arrayList[startIndex] = arrayList[minIndex];
        arrayList[minIndex] = temp;
        selectionSort (arrayList, startIndex + 1)
end selectionSort
```

Selection Sort Cost (N):

Cost (N) = cost of swapping + cost of sorting (n-1) = n-1 + Cost (N-1)

Characteristic Polynomial Approach:

Number of comparisons when size of list is 1 = 0 is C(1) = 0
Number of comparisons when size of list is 2 = 1 is C(2) = 1

T(n) = (n - 1) + C(n - 1) --------------------- (1)
T(n - 1) = (n - 1) - 1 + C(n-2) ------------- (2)

Subtracting (2) from (1)
C(n) - C(n-1)
= n - 1 - n + 2 + C(n-1) - C(n-2)
= C(n) - C(n-1) -C(n-1) + C(n-2) + 1 = 0
= C(n) - 2C(n-1) + C(n-2) + 1
= - 2X + 1 = 0
x = 1

So from characteristic polynomial, cost is C(n) = 1 * (n - 1) = n - 1

Q04.
At first let's see the algorithm of modified Insertion Sort (with Binary Search).

```
for i = 1 to N do
    newElement = list[i]
    location = i - 1
   left = 1
    right = location

    while (left < right) do
        middle = (left + right) / 2
        if (newElement >= list[middle])
        left = middle + 1
        else
            right = middle
    end while
```

```
        list [location + 1] = newElement
    end for
```

If we consider the list as a binary tree, we see that there are i comparisons needed to find the $2^{i-1}$ elements on level I of the tree. For a list N = $2^k - 1$ elements, there are k levels in the binary tree.

There is exactly, *n*-1 iterations are done in the for loop (outer loop). In each iteration, a binary search is done to determine the position at which to do the insertion. In the $i^{th}$ iteration of the outer loop, the binary search considers array positions 0 to i (for 1 <= I <= n). The running time for the binary search in the $i^{th}$ iteration is $\log_2(i+1)$. Once the correct position is found, at most i swaps are needed to insert the element in its place.

On Average case modified insertion sort uses a fewer comparisons than the original insertion sort algorithm.

*average case:* each element is about halfway in order.

$$\sum_{i=1}^{N-1} \frac{i}{2} = \frac{1}{2}\left(1+2+3...+(N-1)\right) = \frac{(N-1)N}{4}$$
$$= O\left(N^2\right)$$