1)

a)

Pseudo code of algorithm is:

Selection\_Sort(Arr)

1 for i=1 to Arr.length –1

2 minimumIndex = i

3 for j = i + 1 to Arr.length

4 if Arr[j] < Arr[minimumIndex]

5 minimumIndex = j

6 temp = Arr[i]

7 Arr[i] = Arr[minimumIndex]

8 Arr[minimumIndex] = temp

b)

The first loop variant is the subarray Arr[1 …. i – 1] has i-1 elements which are the smallest elements from the actual array at every iteration start of outer for loop.

The second loop variant is at the beginning of inside for loop, Arr[minimumIndex] must be the smallest value from Arr[i…. j – 1].

Proof:

Lets assume that we have a array A and the elements of array are [4,6,1,8]. Starting index is 1 and A[minimumIndex] = 4. According to first loop invariant, Arr[i-1] subarray has i-1 elements in that case Arr[0] has 0 small elements so loop invariant is true. Second loop invariant states that Arr[1] is the smallest value from Arr[1] and this is also true. At the end of first iteration array becomes [1,4,6,8].

Now i equals to 2 and A[2] = 4. The first rule states Arr[1 .. 2] has 1 smallest element from complete array and it is true. Second statement is also true because of the same reasons which I explained above.

As a summary, we can conclude that there are 2 loop invariants which I described at the beginning of part b and I proved that they are correct by giving an example.

c)

In this part we should find the worst and best case runtime for selection sort algorithm. In this algorithm n-1 element will be compared with n-i-1 elements and it doesn’t matter if it is the worst case or best case because if we consider the worst case, the algorithm go into the if section but still the program run because the if block doesn’t affect that much the complete run time due to there are two nested for loops and because of them the **best ,worst and average runtime of the selection sort algorithm is .**

2)

a)

as we know insertion sort time complexity is because if we have n element, we must shift n elements for n times in worst case and as average we say that insertion sort runtime for n element is . When we look at the pseudocode , we must understand what does the code between line 5 and 7. First operation of this code snippet is to shifting elements which are greater than value of key to the end direction of the array. But before shifting these elements first it needs to find the position of key by searching it linearly from the end of the array to the start of the array. If we change this code with binary-search method finding the key element runtime will be logn , and for the complete array , I mean for n elements , finding the proper key array it will cost nlogn time to accomplish that objective. But even if we find the proper key, we must shift elements which greater than key and **as a result even if we use binary-search ,still the run time of the insertion sort will be .**

b)

with doubly linked list, running time of shifting process decreased to from with change int pointers of the nodes of doubly linked list. But on the other hand searching will take as normal insertion sort takes. We cannot use binary search algorithm with running time of logn when we use doubly linked list instead of normal array. **Therefore runtime is still .**

3)

As question stated, we can write insertion sort as a recursive method. To accomplish that, there are two objectives. We will call the runtime of our method as T(n).

First objective is sorting the A[1… n-1] sub array.

Second objective is inserting the A[n] element to correct position in A[1… n-1] subarray which is already sorted in first objective.

As a base case we can say that if n=1, A array is empty because A[1-1] = A[0] and in that case applying second objective rule should take constant time which is time and firs objective wont take any time.

If 1<n statement is true, we will call recursive method for n-1 and about applying 2 objective, it should take time because if we find the correct position for A[n] element in A[1… n-1] sub array, we probably but not certainly shift n-1 element in A[1… n-1] sub array so run time becomes for second objective.

As a result of these calculations, final equation is: