Lab 5 - Proofs

In this lab, we are going to work to become comfortable with the CLRS notation and answering basic problems about algorithmic performance and correctness

Problem 1: Insertion Sort, Descending order

Using CLRS's notation, rewrite pseudocode for Insertion Sort so that it sorts in descending order:

There is an array A[1....n]

The number *n* of elements in *A* is denoted by *A.length.*

INSERTION-SORT(A)

for j = 2 **to** *A.length*

    Key = A[*j*]

    // Insert A[*j*] *into the sorted reverse A*[*j-1...1*]

    i = j - i

    While *i > 0 and A*[*i*] *<key*

        A[*i+1*] *=A*[*i*]

        i = i - 1

    A[*i + 1*] *= key*

Problem 2: Linear Search pseudocode

We're going to define the searching problem in CLRS terms.

Input: A sequence of n numbers A = $\langle a_1, a_2, \ldots a_n \rangle$

Output: An index i such that $v = A[i]$ or the special value NIL if v does not appear in A.

Write pseudocode in CLRS style for linear search, which scans through the sequence, looking for v. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

Linear search(A)

Pseudocode

for *i = 1* to *A.length*

  If *A*[*i*] *== v*

  return i

return NIL

A loop invariant

1) Invariant: we have traversed *i-1* elements of array *A.The subarray A =[1...i -1]* does not contain the target value *v.*

2) Initialization: *Before the first iteration of the loop, i =1 and t*he subarray *A* =[*1...i -1*] is empty.

3) Maintenance: For each iteration of the loop, we check *A*[*i*] is not the target number, showing that each iteration maintains the loop invariant.

4) Termination: If we terminate the loop with returning index *i*, it is when we already find an element *A*[i] equals to the value *v.* In this case, our algorithm is correct.

If we terminate the loop without returning index *i*, which means there is not an element of *A*[*1...n*] equals to the value *v.* And we return NIL as required. Hence, the algorithm is correct.

Problem 3: Selection Sort

In CLRS notation, write pseudo for selection sort. Assume that selection sort works this way: It sorts *n* numbers stored in array *A* by first finding the smallest element of *A* and exchanging it with the first element in *A*. Then it finds the second smallest item in *A* and exchanges it with the second item in *A*. It continues for the first *n-1* elements in *A*.

Write pseudocode for selection sort.

SELECTION-SORT(A)

Function Swap(a, b)

temp = a

a = b

b = temp

for *i = 1* ***to*** *A.length - 1*

key = i

for *j = i + 1* ***to*** *A.length*

If *A* [*j*] < *A* [*key*]

key = j //record the index of minimum

Swap *A*[*key*] *and A*[*i*]

What loop invariant does this algorithm maintain?

ANS Invariant: At the start of each iteration of the for loop , the *A*[*1...i -1*] we have traversed has *i-1* elements in sequence.

Why does it need to run for only the first *n-1* elements, instead of for all *n* elements?

ANS At the first iteration, we take the first item *i* to be the smallest one, and then comparing it with the rest element of the array. After *n-1* elements have been sorted. The last one element is the biggest one, so we do not need to run for all *n* elements.

Bonus: Problem 4: Why we never consider best case

How can we modify any sorting algorithm to have a good best case running time, returning a definitely sorted array in O(n)? Remember that, in the best case, you can assume the data is in the most beneficial format that you need for your solution to work.

ANS

We can define a *flag* in for loop of the sort algorithm, and initialize the value of *flag equals to 0(which means true).* Also under the swap function line, we evaluate the *flag equals to 1(which means false)*.

If we have a sequence array, the swap function will never be operated, and the flag will still equals 0(true).

Add the statement *if flag = 0,* break the for loop. The big-Oh complexity of best case is *O(n).*