Assignment 1

Mingyuan Ba
SID: 530157791

March 9, 2024

Problem 1

a)

The number of iterations is

$$(n-1)+(n-2)+\cdots+1+0=\frac{n(n-1)}{2}$$

which is bounded by n^2, For each iteration corresponding to indices i, j, we only need to perform a single comparison operation, so the time complexity per iteration is $O(1)$. Thus , the overall time complexity is $O(n^2)$.

b)

Assume for simplicity that n is even. To lowerbound the running time, consider only comparisons made during the first half of its execution. Since this is part of the full execution, analyzing only this part gives a lower bound on the total running time. The main observation we need is that for each of the considered iterations, we make at least $\frac{n}{2}$ comparisons, allow us to lower bound the total number of comparisons made:

$$\sum_{i=0}^{n-1}(n-i-1) \geq \sum_{i=0}^{\frac{n}{2}-1} \frac{n}{2}=\frac{n^2}{4}=\Omega(n^2)$$

Problem 2

Sum()

When a new element get pushed or poped, the sum need to be updated. The **Sum** operation return the value of variable **sum**, which takes O(1) time.
We modified the push and pop operations. Adding the new element to the sum takes O(1) time, so push still runs in O(1) time. Similarly, subtracting the removed element from the sum takes O(1) time, so pop still runs in O(1) time.

```
function NewPush(e)
    sum ← sum + e
    push(e)
end function
```

```
function NewPop
    e ← pop
    sum ← sum − e
end function
```

```
function Sum
    return sum
end function
```

Problem 3

a)

```
function A(B, m)
    n ← sizeof(B)
    j ← n − 1
    i ← 0
    counts ← 0
    while i < j do
        if B[i] + B[j] ≥ m then
            counts ← counts + (j − i)
            j ← j − 1
        else
            i ← i + 1
        end if
    end while
    return counts
end function
```

b)

Theorem: For B, m given above,the algorithm A should correctly calculate the number of index pairs i, j such that $B[i] + B[j] \geq m$ for all $i < j$.

Base Case:
$n = 2$,B has only 2 element $B[0], B[1]$. The algorithm A compares $B[0] + B[1]$ with m. If $B[0] + B[1] \geq m$, A sets *counts* to 1 which is correct. Else, there is no valid index pair.

Inductive Step:
Assume that for any array B of length k algorithm A works correctly. We need to prove that for an array B of length $k + 1$, the algorithm A can also correctly calculate the number of satisfying index pairs.
Proof:

The algorithm A starts checking from $i = 0$ and $j = k$.

- If $B[0] + B[k] \geq m$, given the sorted property, all elements between $B[0]$ and $B[k]$ also sum up to be greater than or equal to m. $k - i$ correctly counts the number of pairs with $B[k]$.Then, algorithm decrease j by 1(which is $k - 1$ now). Based on the assumption, the algorithm has already calculated all valid index pairs within the indexes from 0 to $k-1$.Thus, algorithm finds all valid index pairs for an array B of length $k + 1$.

- If $B[0]+B[k] < m$, the algorithm increments i by 1 in a loop until $B[i]+B[k] \geq m$ is satisfied. At this point, $k - i$ is added to the counts, or the loop continues until $i = j$ (indicating there are no valid index pairs). In both cases, we have found all valid pairs involving $B[k]$. Based on the assumption, we have found all valid pair involving B[k-1]. Thus the algorithm finds all valid index pairs for an array B of length $k + 1$.

Conclusion

Therefore, by induction, we conclude that for any sorted array B of length n and any positive integer m, the algorithm $A(B, m)$ correctly calculates the number of index pairs (i, j) such that $B[i] + B[j] \geq m$.

c)

Line 2-5, 7,11,14 all consist of assignments, simple comparisons, simple math operations and the return statements return booleans, all of which takes O(1) time.Each iteration of the while loop on *line 6-13* perform O(1) time operations.And either i is incremented by 1(*line 11*),or j is decremented by 1(*line 9*),Hence, each iteration decreases the distance between i and j by 1.Given that the initial distance between i and j is $n - 1$ (where n is the length of the array B), the loop can run at most $n - 1$ times.Thus,the loop takes at most $O(1) \cdot O(n)$ or $O(n)$ time.. Since we saw that the operations outside the loop all take constant time, the loop dominates the running time, which comes down to $O(n)$ time in total, as required.