Recitation- Week 2

Michael Levet

August 29, 2020

1 Loop Invariants

Problem 1. Determine the *useful* loop invariant for the following algorithm.

```
LinearSearch(A[1, ..., n], key):
   ret = -1
   for i = 1 to n:
        if A[i] == key:
           ret = i
   return ret
```

Problem 2. Recall the *Selection Sort* algorithm.

```
SelectionSort(A[1, ..., n]): //A is an array of numbers
   for i = 1 to n-2:
        minIndex = i
        for j = i + 1 to n-1:
             if A[j] < A[minIndex]:
                minIndex = i
        swap A[i] and A[minIndex]
```

Do the following.

(a) Determine the *useful* loop invariant for the inner loop.

(b) Determine the *useful* loop invariant for the outer loop.

2 Algorithm Analysis

Problem 3. Analyze the *worst-case* runtime of the following algorithm. Clearly derive the runtime complexity function $T(n)$ for this algorithm. Avoid heuristic arguments from 2270/2824 such as multiplying the complexities of nested loops.

```
add(int n): // assume n >= 0
    sum = 0
    for i = 1; i < n; i = i + 1:
         sum = sum + i
    return sum
```

Problem 4. Analyze the *worst-case* runtime of the following algorithm. Clearly derive the runtime complexity function $T(n)$ for this algorithm. Avoid heuristic arguments from 2270/2824 such as multiplying the complexities of nested loops.

```
addFirstHalf(int n): // assume n >= 0
    sum = 0
    for i = 1; i < n/2; i = i + 1:
         sum = sum + i
    return sum
```

Problem 5. Analyze the *worst-case* runtime of the following algorithm. Clearly derive the runtime complexity function $T(n)$ for this algorithm. Avoid heuristic arguments from 2270/2824 such as multiplying the complexities of nested loops.

```
addMultiplesOf3(int n): // assume n >= 0
    sum = 0
    for i = 3; i < n; i = i + 3:
         sum = sum + i
    return sum
```

Problem 6. Analyze the *worst-case* runtime of the following algorithm. Clearly derive the runtime complexity function $T(n)$ for this algorithm. Avoid heuristic arguments from 2270/2824 such as multiplying the complexities of nested loops.

```
addFirstN(int n): // assume n >= 0
    sum = 0
    for i = 1; i < n; i = i * 2:
         sum = sum + i
    return sum
```

Problem 7. Analyze the *worst-case* runtime of the following algorithm. Clearly derive the runtime complexity function $T(n)$ for this algorithm. Avoid heuristic arguments from 2270/2824 such as multiplying the complexities of nested loops.

```
sum(int n): // assume n >= 0
    sum = 0
    for i = 1; i < n; i = i + 1:
         for j = 1; j < n; j = j + 3
              sum = sum + i
    return sum
```

Problem 8. Analyze the *worst-case* runtime of the following algorithm. Clearly derive the runtime complexity function $T(n)$ for this algorithm. Avoid heuristic arguments from 2270/2824 such as multiplying the complexities of nested loops.

```
sum(int n): // assume n >= 0
    sum = 0
    for i = 1; i < n; i = i + 1:
         for j = 1; j < n; j = j + 1
              for k = 1; k < j; k = k + 1
                   sum = sum + i
    return sum
```