cs361 Midterm Review

Ryan Scherbarth

February 2024

1 Midterm Content

- Finding the running time of a given program/algorithm
- Loop invariant (initialization, maintenance, and termination). The specification should be precise and reflect the correctness of the whole algorithm.
- Solving recurrence relations
- Inductive proofs
- The algorithms that we have learned along with their best, worst, and average running times.
- Linked lists and the operations on them.

2 Running Time

The **running time** of an algorithm on a particular input is the number of instructions and data accesses executed.

2.1 Insertion sort

```
Insertion-Sort(A, n)
  for i = 2 to n
    key = A[i]
    // Insert A[i] into sorted A[1:i-1]
    j = i - 1
    while j > 0 and A[j] > key
      A[j+1] = A[j]
      j = j - 1
    A[j+1] = key
```

Cost	Label
c_1	n
c_2	$n-1$
0	$n-1$
c_4	$n-1$
c_5	$\sum_{i=2}^{n} t_i$
c_6	$\sum_{i=2}^{n}(t_i-1)$
c_7	$\sum_{i=2}^{n}(t_i-1)$
c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1)$$

The best/worst case running times can depend on the type of input given. In the case of insertion sort, the best case comes when we pass in an already sorted array. In this case, lines 5-7 immediately exit upon the test on line 5. Therefore, We can see the best case running time is given as:

2.1.1 Best Case

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

Therefore, the best case running time for insertion sort is linear; $O(n)$

2.1.2 Worst Case

We get the worst case running time for insertion sort when we are given an array in reverse order. While the beginning timing for the outer for loop remains the same, we need to be careful of the inner for loop, which, rather than exiting on the first iteration each time, will now take the maximum iterations each time we reach it. Here, we would calculate:

$$\sum_{i=2}^{n} i = \sum_{i=1}^{n} (i) - 1$$

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

(Corresponding to c_5)

$$\sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i$$

$$\frac{n(n-1)}{2}$$

(Corresponding to c_6, c_7)

$$T(n) = c_1(n)+c_2(n-1)+c_4(n-1)+c_5(\frac{n(n+1)}{2}-1)+c_6(\frac{2(n-1)}{2})+c_7(\frac{n(n-1)}{2})+c_8(n-1)$$

$$= \frac{c_5+c_6+c_7}{2}n^2 + (c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8)n - (c_2+c_4+c_5+c_8)$$

Therefore, the worst case running time for insertion sort is quadratic; $O(n^2)$ Hello World!

2.1.3 Order of Growth

Big Theta notation; $\theta(n)$, is used to represent the rate of growth of the time with respect to the problem size, n. In the case of insertion sort, we would simplify our running times to the largest order n values. This gives us a best case running time of $\theta(n)$, and a worst case running time of $\theta(n^2)$.

3 Loop Invariants

Loop invariants consist of three parts;

- **Initialization**
- **Maintenance**
- **Termination**

3.1 Loop invariant for insertion sort

Given the insertion sort algorithm:

```
Insertion-Sort(A, n)
  for i = 2 to n
    key = A[i]
    // Insert A[i] into sorted A[1:i-1]
    j = i - 1
    while j > 0 and A[j] > key
      A[j+1] = A[j]
      j = j - 1
    A[j+1] = key
```

3.1.1 Initialization

Before the loop starts, when $i = 0$, the subarray $A[0..i-1]$ consists of the sorted elements, and the subarray $A[i+1..n-1]$ consists of the remaining unsorted elements. It is trivially true that $A[0..i-1]$ will be sorted at this point since this subarray doesn't contain any elements yet.

3.1.2 Maintenance

At the start of each iteration of the loop, the element at index i is inserted into its correct position within the sorted subarray $A[0..i-1]$. After this insertion, the subarray $A[0..i]$ will be sorted, and the subarray $A[i+1..n-1]$ will remain unsorted.

3.1.3 Termination

The loop terminates when i reaches the final iteration of the outer for loop, $i = n$. At this point, the entire array is sorted, since we have already proven that the subarray $A[0..i]$ is properly sorted, and we now have $A[0..i = n]$.

4 Recurrance Relations