CS 624: Notes 02

Ryan Culpepper

September 12, 2022

These notes contain an outline of what I said in lecture (but only an outline), and they also contain interactive questions and exercises. The corresponding slides are in `slides01.pdf`.

1 Administrative

- Course web page updated
- Lecture 01 video on Blackboard
- Register for gradescope with umb.edu address
  Automatically added to CS 624 page
- Register for Piazza
  Add yourself to the course piazza page
- Zoom: make sure to use a umb.edu account!
- Homework 01 will go out this evening
  covers material through Wednesday (9/14)
  due week from Wednesday (9/21)

2 Continuing from Lecture 01

In Lecture 01, we saw the InsertionSort algorithm and started reasoning about its correctness.

2.1 Reminder: Slide 11: Insertion Sort

```
InsertionSort(A) :=

for j ← 2 to length[A] do
  key ← A[j]
  i ← j - 1
  // Insert A[j] into sorted sequence A[1..j-1]
  while i > 0 and A[i] > key do
```

```
    A[i+1] ← A[i]
    i ← i - 1
  end while
  A[i + 1] ← key
end for
```

2.2 Reminder: Slide 14: Loop Invariant

Loop invariant:

> The numbers in `A[1 .. (j-1)]` are in sorted order, and they are a permutation of the original `A[1 .. (j-1)]`.

3 Slide 15

Parts to proof by loop invariant:

- Initialization
- Maintenence
- Termination

Similar to induction.

(skip to slide 18)

4 Slide 20

Example:

$$\langle 2, 4, 5, 6, 7, 9 \rangle$$

Discuss:

Best-case analysis is good!

?

Best-case analysis is bad!

?

5 Slide 22

What is t_j?

6 Slide 25

Discuss:

Average-case analysis is good!

- best case might be misleading

Average-case analysis is bad!

- your inputs might not be ”average”

Average vs worst:

> If you always plan for the worst, then all of your surprises will be happy ones.

vs

> If you are too pessimistic, you might give up, over-budget, etc.

We solve “intractably hard” problems all the time! Examples:

- scheduling, solving constraints
- NP-complete problems like SAT-solvers
- type inference (polymorphic) like ML, Haskell, etc

Precision:

What does “average-case” mean?

Average over *what distribution* of inputs?

In slides 25–34, *uniform distribution* over permutations (assuming all elements in sequence are distinct).

7 Slide 26

An *inversion* is when a little number follows a big number in the array.

An element may participate in multiple inversions.

$$\langle 5, 1, 6, 4, 7, 2 \rangle$$

How many inversions does 4 participate in?

3 (2 on the left, 1 on the right)

If we shuffle the numbers before 4, does it change the answer?

no

8 Slide 27: What do inversions have to do with the run time?

If the array has N inversions of the form $(\cdot, a_j)$, then the inner loop will execute N times for that value of j.

9 Slide 29: What is the average number of inversions?

The average number of inversions (over a uniform distribution of permutations of distinct elements of an array) is

$$\frac{\textit{total inversions over all permutations}}{\textit{number of permutations}}$$

How can we count the total number of inversions?

Related: What is the sum of the numbers 1 .. 100?

9.1 Summation solution

First we calculate *twice* the sum, by pairing off elements with elements in the reversed sequence.

```
1   + 2  + 3  + ... + 100 +
100 + 99 + 98 + ... + 1

=

101 + 101 + 101 + ... + 101     <-- 100 elements
=
101 × 100
=
10100
```

Then divide by 2:

$$5050$$

9.2 Counting inversions

How can we count the total number of inversions?
First we count them twice!

What is the number of inversions in a permutation plus the number of inversions in the reverse permutation? (Assuming all elements are distinct.)

For each pair of indexes $i < j$, either

- $A_p[i] > A_p[j]$ — an inversion (i, j) in the forward permutation, or
- $A_p[i] < A_p[j]$ — there is an inversion in the reverse permutation; not at (i, j), but at $(L + 1 - j, L + 1 - i)$

10 Slide 35: Merge Sort

11 Slide 37: Merge

`Merge(A, p, q, r)`

Preconditions:

- $1 \le p \le q \le r \le$ `length[A]`, and
- the array sections `A[p..q]` and `A[q+1..r]` are sorted

Postconditions:

- `A[p..r]` is sorted, and
- `A[p..r]` is a permutation of the original contents of `A[p..r]`

12 Slide 38: Loop Invariant for Merge

At the start of each iteration of the `for` loop on k, the subarray `A[p..k-1]` contains the $k - p$ smallest elements of `L[1..n`$_1$`+1]` and `R[1..n`$_2$`+1]`, in sorted order.

Moreover, `L[i]` and `R[j]` are the smallest elements of their arrays that have not been copied back into A. (Or ∞.)

13 Slide 41

Typo in slide, should be

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ 2T(n/2) + \underline{c}n & \text{otherwise} \end{cases}$$

14 Slide 42

(See Figure 2.5 in the textbook.)

What is $T(n)$?

```
T(n)
```

We don't know, but (if $n \neq 1$), we can unfold it one level:

```
            c*n +
      T(n/2) +  T(n/2)
```

but what is $T(n/2)$? We don't know, but (if $n/2 \neq 1$), we can unfold another level:

```
                           c*n +
         c*n/2              +            c*n/2 +
  c*T(n/4) + c*T(n/4)       +   c*T(n/4)  + c*T(n/4)
```

How many levels can we unfold it?

$log_2 n$

What happens when we get to the bottom?

dn (more or less)

What is the sum of all costs in the tree?

$$cn \log_2 n + dn$$

Note the imprecision! In fact, different branches are likely to reach the base case at slightly different times. Why is that okay?

15 Next: Asymptotic Analysis

In the next lecture, we'll talk about about how to be rigorously imprecise!