

Algorithms, Spring 2004
Solution for Homeworks 3, 4, 5

Homework 3 Solutions

Solution: 1a Binary search takes $O(\log n)$ for n elements. In case of insertion sort, the binary search to find the place for the element to insert will take $\log 1, \log 2, \log 3, \dots, \log(n-1)$. The total time will be sum of these log terms which tends to $O(n \log n)$.

Solution: 1b Yes because insertion sort is a comparison based sorting algorithm.

Solution 2: There are many ways to do this problem. Given here is one that uses a heap. Let the d sequences be s_1, \dots, s_d . Build a heap on the items $1, \dots, d$ with keys $s_1[1], \dots, s_d[1]$, respectively, and delete $s_i[1]$ from s_i , for $i = 1, \dots, d$. While not all sequences are empty, we do the following: extract the minimum item from the heap and output its key. If the minimum item is i , we insert i with the first element in s_i as key onto the heap and delete that element from s_i . If s_i is empty then arbitrarily pick another nonempty sequence for this operation. Continue until all sequences are empty. Building the heap takes $O(d) = O(n)$ time. Extracting the minimum and insertion takes $O(\log d)$ time. Since every element in s_1, \dots, s_d is considered only once, the total time taken is $O(n \log d)$.

Homework 4 Solutions

solution 1. Consider the innermost loop: the output statement is executed

$$\sum_{k=i}^{j-1} 1 = (j-1) - i + 1 = j - i = (i + L - 1) - i = L - 1$$

times. Summing over one full pass through the middle loop we have

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

The outermost loop will range from $L = 2$ to n . Summing over one full pass of outer loop we have

$$T(n) = \sum_{L=2}^n (n(L-1) - (L-1)^2) = n \sum_{L=2}^n (L-1) - \sum_{L=2}^n (L-1)^2 = n \sum_{l=1}^{n-1} l - \sum_{l=1}^{n-1} l^2.$$

Here we made the change of variables $l = L - 1$. The first sum is an arithmetic series and the second is a quadratic series. Applying standard formulas and simplifying we have

$$T(n) = \frac{n^3 - n}{6}$$

solution: 2 We will express $C[i_0, i_1, j_0, j_1]$ in terms of the cost of sub-rectangles. There are two cases for general rectangles. First, if sub array $A[i_0..i_1, j_0..j_1]$ is all of one color, then we do not need to perform

any subdivision and so the cost is 0. (Note that this takes care of the basis case, since a single pixel is a rectangle of width one, and each pixel has only one color.) Otherwise, we need to partition it. There are two subcases. If the partition is horizontal, then there is an index k , $i_0 \leq k < i_1$ such that the array is broken up as $A[i_0..k, j_0..j_1]$ and $A[k+1..i_1, j_0..j_1]$. On the other hand, if the partition is vertical, then there is an index k , $j_0 \leq k < j_1$, such that the array is broken up as $A[i_0..i_1, j_0..k]$ and $A[i_0..i_1, k+1..j_1]$. This costs 1 cut plus the optimum cost of subdividing the two sub matrices. Since we do not know whether to use a vertical or horizontal cut, and we do not know where to make the cut, we will just try all the possibilities. Combining all this gives the following rule

$$C[i_0, i_1, j_0, j_1] = \begin{cases} 0 & \text{if } A[i_0..i_1, j_0..j_1] \text{ is of one color} \\ 1 + \min(H, V) & \text{otherwise.} \end{cases}$$

where

$$H = \min_{i_0 \leq k < i_1} (C[i_0, k, j_0, j_1] + C[k+1, i_1, j_0, j_1])$$

$$V = \min_{j_0 \leq k < j_1} (C[i_0, i_1, j_0, k] + C[i_0, i_1, k+1, j_1])$$

The final output is the cost of segmenting the entire image, that is, $C[1, m, 1, n]$.

solution 3. Consider matrices $A_1 \times A_2 \times A_3$, defined by the sequence $p[0..3] = \{1, 1, 2, 3\}$ (where A_i is a $p_{i-1} \times p_i$ matrix). The greedy algorithm selects the order $A_1(A_2A_3)$ (since the $p[1] < p[2]$), and this results in a total cost of $(1 \times 2 \times 3) + (1 \times 1 \times 3) = 9$. In contrast, the optimal order is $(A_1A_2)A_3$, at a total cost of $(1 \times 1 \times 2) + (1 \times 2 \times 3) = 8$, which is superior. Intuitively, the problem with the greedy approach is that it only considers the effect of the middle dimensions (either $p[1]$ or $p[2]$), ignoring the effects of the outer dimensions ($p[0]$ and $p[3]$). In the example above, we make $p[3]$ much larger than $p[0]$, and this is the difference between the two.

Homework 5 Solutions

Solution 1: The obvious greedy algorithm is to drive as far as possible (but without running out of gas) before filling up. Let n be the number of miles miles that the car can travel after fueling and let x_1, x_2, \dots, x_m denote the miles to the m petrol pumps along the professor's route, assumed to be sorted in increasing order. We make the assumption that $x_i - x_{i-1} \leq n$, for all i , for otherwise it is impossible to make the journey without running out of petrol. We will also assume that the professor starts with a full tank. To simplify notation, we assume that there is an extra stop at $x_{m+1} = \infty$. The variable A holds the final fillup schedule of stops, and the variable `prevStop` holds the mile indicator of the last stop that we made.

```
gasStop(x[1..m], n)
{
    prevStop = 0; // location of previous stop
```

```

A = {};          // start with empty list
for i = 1 to m do // passing petrol pump i
{
    if (prevStop + n < x[i+1]) // can't make it to next stop
    {
        prevStop = x[i]; // then stop here
        append i to A;
    }
}
return A;
}

```

solution 2. Strong Components.

