

CS 430 - Homework 1, Solution sketches (without some proofs)

Problem 1 Order the following list of functions by the big-Oh notation. Group together (for example, by underlining) those functions that are big-Theta of one another.

$$\begin{array}{cccccc}
 6n \log n & 2^{100} & \log \log n & \log^2 n & 2^{\log n} & \\
 2^{2^n} & \lceil \sqrt{n} \rceil & n^{0.01} & 1/n & 4n^{3/2} & \\
 3n^{0.5} & 5n & \lfloor 2n \log^2 n \rfloor & 2^n & n \log_4 n & \\
 4^n & n^3 & n^2 \log n & 4^{\log n} & \sqrt{\log n} &
 \end{array}$$

Hint: When in doubt about two functions $f(n)$ and $g(n)$, consider $\log f(n)$ and $\log g(n)$ or $2^{f(n)}$ and $2^{g(n)}$.

Solution Sketch:

(1) $\frac{1}{n} : \theta(\frac{1}{n})$

(2) $2^{100} : \theta(1)$

(3) $\log \log n : \theta(\log \log n)$

(4) $\sqrt{\log n} : \theta((\log n)^{\frac{1}{2}})$

(5) $\log^2 n : \theta(\log^2 n)$

(6) $n^{0.01} : \theta(n^{0.01})$

(7) $\lceil \sqrt{n} \rceil, 3n^{0.5} : \theta(n^{0.5})$

(8) $2^{\log n}, 5n : \theta(n)$

(9) $n \log_4 n, 6n \log n : \theta(n \log n)$

(10) $\lfloor 2n \log^2 n \rfloor : \theta(n \log^2 n)$

(11) $4n^{\frac{3}{2}} : \theta(n^{\frac{3}{2}})$

(12) $4^{\log n} : \theta(n^2)$

(13) $n^2 \log n : \theta(n^2 \log n)$

(14) $n^3 : \theta(n^3)$

(15) $2^n : \theta(2^n)$

(16) $4^n : \theta(4^n)$

(17) $2^{2^n} : \theta(2^{2^n})$

Problem 2 Describe a method for finding both the minimum and maximum of n elements using fewer than $3n/2$ comparisons between elements.

Hint: First construct a group of candidate minimums and a group of candidate maximums.

Note that only the number of comparisons between elements matter, not the running time. Present pseudocode, count the number of comparisons between elements done in the worst case, and argue your solution is correct.

Solution Sketch:

The array is indexed from 1 to n , and the positions of the minimum and maximum element are returned.

```
FIND-MIN-MAX(A)
1  n ← length[A]
2  for i ← 1 to ⌊n/2⌋
3    if A[2i-1] ≤ A[2i]
4      then min-group[i] ← 2i-1
5         max-group[i] ← 2i
6    else min-group[i] ← 2i
7         max-group[i] ← 2i-1
8  min ← min-group[1]
9  max ← max-group[1]
10 for i ← 2 to ⌊n/2⌋
11  if A[min] > A[min-group[i]]
12    then min ← min-group[i]
13  if A[max] < A[max-group[i]]
14    then max ← max-group[i]
15 if n is odd
16  if A[n] < A[min]
17    then min = n
18  if A[n] > A[max]
19    then max = n
20 return (min, max)
```

In the lines 1-7, the n numbers are separated into two groups - the maximum group and the minimum group. Here the number of comparisons are $n/2$ if n is even, or $\lfloor n/2 \rfloor$ if n is odd.

The lines 8-14 use linear search to find minimum and maximum number in these two groups. The number of comparisons are $n/2 - 1$ for each of them if n is even, or $\lfloor n/2 \rfloor - 1$ if n is odd.

The line 15-19 handles the situation when n is odd. Here two extra comparisons are added.

Total number of comparisons are $3n/2 - 2$ if n is even, or $3*\lfloor n/2 \rfloor$ if n is odd.

Problem 3 An n -degree *polynomial* $p(x)$ is an equation of the form

$$p(x) = \sum_{i=0}^n a_i x^i,$$

where x is a real number and each a_i is a constant.

- a. Describe a simple $O(n^2)$ time method for computing $p(x)$ for a particular value of x , given as input x and the length- $(n + 1)$ array of coefficients A .
- b. Consider now a rewriting of $p(x)$ as

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \dots + x(a_{n-1} + xa_n)\dots))),$$

which is known as *Horner's method*. Using the big-Oh notation, characterize the number of multiplications and additions this method of evaluation uses.

Present pseudocode.

Solution Sketch:

- a. POLYNOMIAL(A, x)

```
1  sum ← a[0]
2  for i ← 1 to n - 1
3    xi ← 1
4    for j ← 0 to i - 1
5      xi ← xi*x
6    sum ← sum + a[i]*xi
7  return (sum)
```

- b. HORNER(A, x)

```
1  sum ← a[n]
2  for i ← n - 1 to 0
3    sum ← sum*x + a[i]
4  return (sum)
```

The number of multiplications and additions is n .

Problem 4 Give pseudocode for the following problem: give an array $A[1 \dots n]$ containing the numbers $1, 2, \dots, n$ and representing a permutation, modify A to represent the next lexicographic permutation.

Analyze running time (which should not exceed $O(n)$) and argue correctness (why the output is lexicographically bigger than the input? why other permutations lexicographically bigger than the input are bigger than the output).

Solution Sketch:

```
NEXT-PERMUTATION(A)
1  n ← length[A]
2  i ← n - 1
3  while i > 0 and A[i] ≥ A[i+1]
4    i ← i - 1
5  if i = 0
6    then return (nil)
7  m ← n
8  while A[i] ≥ A[m]
9    m ← m - 1
10 tmp ← A[i]
11 A[i] ← A[m]
12 A[m] ← tmp
13 j ← i+1
14 k ← n
15 while j < k
16   tmp ← A[j]
17   A[j] ← A[k]
18   A[k] ← tmp
19   j ← j + 1
20   k ← k - 1
21 return (A)
```

Lines 1-4 find the position i in the array such that all the elements after i in the array are arranged in descending order. This indicates that there are no more permutations beginning with $A[1] \dots A[i]$. If i is zero at this point, then entire array is arranged in descending order and there is no permutation that is larger as the largest numbers are in the most significant positions. In this case Lines 5-6 return nil.

Lines 7-9 scan the elements in the array after position i to find the smallest number that is larger than $A[i]$. Since $A[i] < A[i+1]$, we know that such a number exists. This number $A[m]$ is then swapped with $A[i]$ in Lines 10-12, so that now $A[1] \dots A[i]$ is larger than it was before.

At this point the later part of the array, $A[i+1] \dots A[n]$, is still arranged in descending order. Swapping the values in $A[i]$ and $A[m]$ did not affect the ordering since $A[m]$ is the smallest value larger than $A[i]$. Hence $A[m-1] \geq A[m] > A[i] \geq A[m+1]$.

To avoid skipping permutations, the order of the values in $A[i+1] \dots A[n]$ needs to be changed from descending to ascending. Lines 13-20 do this by simply reversing the values in $A[i+1] \dots A[n]$. Numbers arranged in ascending order produce a permutation that has the least value since the

smallest values are in the most significant positions. This ensures that A contains the smallest possible permutation starting with the numbers $A[1] \dots A[i]$. However, since the value in $A[i]$ has been replaced with a larger value, A now has a permutation that is larger than it did at the start of the algorithm.

Finding $A[i]$ and $A[m]$ are both $O(n)$ tasks as is reversing the values in $A[i+1] \dots A[n]$. Hence this algorithm is $O(n)$.

- Problem 5**
1. Draw the 11-item hash table resulting from hashing the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, using the hash function $h(i) = (2i + 5) \bmod 11$ and assuming collisions are handled by chaining.
 2. What is the result of the previous exercise, assuming collisions are handled by linear probing?
 3. Show the result of Exercise 1 above in this problem, assuming collisions are handled by quadratic probing, up to the point where the method fails because no empty slot is found.
 4. What is the result of Exercise 1 above in this problem, assuming collisions are handled by double hashing using a secondary hash function $h'(k) = 7 - (k \bmod 7)$?

Solution

(1)

0	1	2	3	4	5	6	7	8	9	10
	20			16	44	94	12		13	
				5	88	39	23			
					11					

(2)

0	1	2	3	4	5	6	7	8	9	10
11	39	20	5	16	44	88	12	23	13	94

(3)

0	1	2	3	4	5	6	7	8	9	10
	20	16	11	39	44	88	12	23	13	94

(4)

0	1	2	3	4	5	6	7	8	9	10
11	23	20	16	39	44	94	12	88	13	5