Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dmytro Chibisov
Sandeep Sadanandan

Winter Semester 2007/08
Solution Sheet Model Test
January 16, 2008

Fundamental Algorithms - Surprise Test

Name:	
Email:	

Problem 1 (20 Points)

Design iterative and recursive algorithms to compute $Factorial(n)$. Compare the complexities.

Solution

Iterative algorithm

```
Algorithm Factorial(n)
(* The iterative algorithm for $Factorial(n)$ *)
1.  $returnval \leftarrow 1$
2.  $i \leftarrow 1$
3.  while $i \leq n$
4.          $returnval = returnval * i$
5.          $i = i + 1$
6.  return $returnval$
```

It is easily seen that the number of operations needed for this algorithm is $2 \cdot n$.

Recursive Algorithm

```
Algorithm Factorial(n)
(* The recursive algorithm for $Factorial(n)$ *)
1.  if $n = 0$
2.    then
3.          return 1
4.    else
5.          return $n * Factorial(n - 1)$
```

It is clear that the number of recursive calls will be n. Each call does two operations. Hence the cost is $2 \cdot n$.

Analysis

Both the methods give a complexity of $O(n)$.

Problem 2 (10 Points)

1. Rank the following functions by order of growth (non-decreasing order)
   $n^2, n!, \ln\ln n, 2^{2^n}, e^n, n^3, n\lg n$
2. Give an example of a single nonnegative function $f(n)$ such that for all functions in part 1, $f(n)$ has no relation.

Solution

1. $\ln\ln n < n\ln n < n^2 < n^3 < n! = e^n < 2^{2^n}$
2. $(1+\sin n)\Pi_i g_i(n)$, where g_i are the functions in part 1

Problem 3 (10 Points)

Write down the contents of *Any One* of the following arrays after every step of selection sort until the array is completely sorted.

Assume that the arrays given represent their initial arrangement of the numbers. Also compute the number of operations needed. (Comparison and Swapping are the operations)

1.

12	8	-2	23	5	0

2.

31	17	29	11	7	5	3

Solution

Selection Sort: In selection sort, one finds out the smallest element of the array and swaps that one with the first element of the array. After this step, as the smallest is already in it's correct position, one focuses on the rest of the array. The next smallest element is found out and then swapped with the second element of the array. Now, as the second element is in it's correct postion, the process continues with the rest of the array. And so on, until the whole array is sorted.

For finding out the smallest element, we need to do $O(n)$ comparisons, and we need to do this n times, which gives us a total complexity of $O(n^2)$.

1.

12	8	-2	23	5	0

   The steps are

   (a) Initial State

12	8	-2	23	5	0

(b) After 5 comparisons and 1 swap

-2	8	12	23	5	0

(c) After 4 comparisons and 1 swap

-2	0	12	23	5	8

(d) After 3 comparisons and 1 swap

-2	0	5	23	12	8

(e) After 2 comparisons and 1 swap

-2	0	5	8	12	23

(f) After 1 comparison and 0 swaps

-2	0	5	8	12	23

2.

31	17	29	11	7	5	3

The steps are

1. Initial State

31	17	29	11	7	5	3

2. After 6 comparisons and 1 swap

3	17	29	11	7	5	31

3. After 5 comparisons and 1 swap

3	5	29	11	7	17	31

4. After 4 comparisons and 1 swap

3	5	7	11	29	17	31

5. After 3 comparisons and 0 swap

3	5	7	11	29	17	31

6. After 2 comparisons and 1 swap

3	5	7	11	17	29	31

7. After 1 comparison and 0 swaps

3	5	7	11	17	29	31

Problem 4 (10 Points)

What is *Divide and Conquer*?.

Give an example for a *Divide and Conquer* algorithm.

Solution

The *divide-and-conquer* strategy solves a problem by:

1. Breaking it into subproblems that are themselves smaller instances of the same type of problem
2. Recursively solving these subproblems
3. Appropriately combining their answers

Example: Binary search

The ultimate divide-and-conquer algorithm is, of course, binary search: to find a key k in a large file containing keys $A[1, \dots, n]$ in sorted order, we first compare k with $A[\frac{n}{2}]$, and depending on the result we recurse either on the first half of the file, $A[1, \ldots, \frac{n}{2}]$, or on the second half, $A[\frac{n}{2}+1, \ldots, n]$. The recurrence now is $T(n) = T(\frac{n}{2}) + O(1)$.

Example: Multiplication

This is something which we do daily in our lives. Without taking a paper and pen, calculate the value of 82×76?

Can we do it as $82 \times 76 = 82 \times (75+1) = (80 \times 75) + (2 \times 75) + (82 \times 1) = 6000 + 150 + 82 = 6232$

What we did was using the formula $(a+b) \cdot (c+d) = ac + ad + bc + bd$

Problem 5 (10 Points)

A Binary Tree is a rooted tree in which every node has at most two children. The root node is said to be in level one. The children of the noded at level n are in level $n+1$.

Calculate

1. The maximum number of nodes in level h of a binary tree
2. The maximum number of nodes in a binary tree of h levels.
3. How many nodes does a *complete*[1] binary tree with n leaves have?

Solution

1. The maximum number of nodes in level h of a binary tree

   A level will have maximum number of nodes iff the level above it has maximum nodes and all the nodes have maximum (2) children. So the maximum nodes which can occur in level h is 2 times the maximum of level $h-1$.

   The maximum of level 1 is 1 which is 2^0

   The maximum of level 2 is 2 which is 2^1

   The maximum of level 3 is 4 which is 2^2

   Hence, the maximum of level h will be 2^{h-1}.

2. The maximum number of nodes in a binary tree of h levels.

   A tree of h levels will have maximum nodes iff all the h levels have maximum nodes in each of them. So the total number of nodes will be $1 + 2 + \ldots + 2^{h-1}$ which is equal to $2^h - 1$.

[1]A binary tree is *complete* if all of its vertices have either zero or two children and all the leaves are at levels l and $l-1$

3. How many nodes does a *complete* binary tree with n leaves have?

   If the tree has maximum number of nodes, it's last level has to have maximum number of leaves. If we assume that the last level is h, then from the first part, we know that the maximum number of leaves at level h is 2^{h-1}. So in this case 2^{h-1} equals n.

   And from the part two, we know that the maximum number of nodes in a tree of levels h is $2^h - 1$.

   We can see that $2^h - 1 = 2(2^{h-1}) - 1 = 2 \cdot n - 1$. So the solution is $2 \cdot n - 1$.

   A different and more mathematical approach is given below. If the complete binary tree is of height h, the number of leaves will be between 2^{h-2} and 2^{h-1}. Hence the height could be calculated to be $\lceil \lg n \rceil + 1$. The maximum number of nodes a tree of height h is $2^h - 1$.

   So in this case, it is $2^{\lceil \lg n \rceil + 1} - 1 = 2^{\lceil \lg n \rceil} \cdot 2 - 1$

Problem 6 (10 Points)

Review all the sort algorithms taken in the class. Compare their complexities.

Prove that the lower bound for sorting is $n \lg n$

Solution

Sort	Average	Best	Worst	Remarks
Bubble sort	n^2	n^2	n^2	
Selection sort	n^2	n^2	n^2	
Insertion sort	n^2	n	n^2	In best case, insert requires constant time
Merge sort	$n \lg n$	$n \lg n$	$n \lg n$	
Heap sort	$n \lg n$	$n \lg n$	$n \lg n$	
Quick sort	$n \lg n$	$n \lg n$	n^2	

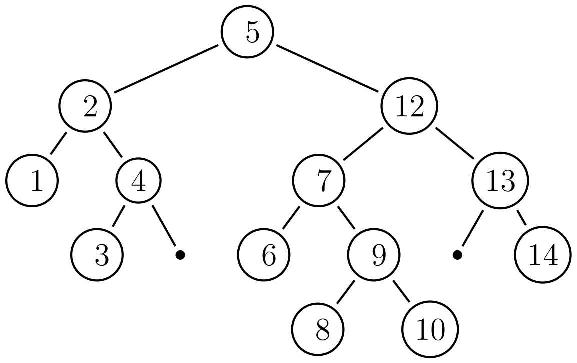
PROOF:

For an input of size n, the decision tree has $n!$ leaves. Which leaves the tree with a height $h \geq \lg(n!)$

$$\begin{aligned} h &\geq \lg(n!) \\ &\geq \lg\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) \\ &= \frac{n}{2}(\lg(n) - 1) \\ &\geq \left(\frac{n}{4}\right)\lg n \end{aligned}$$

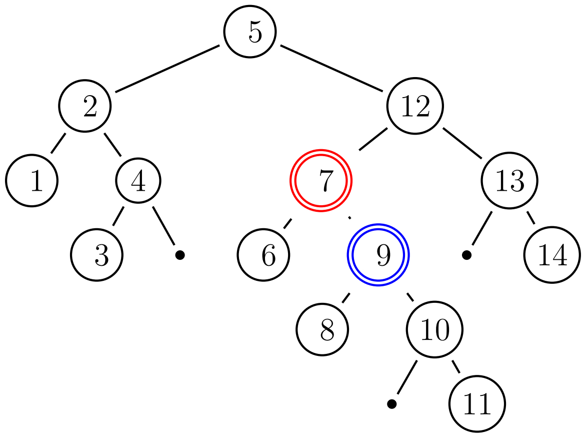
Problem 7

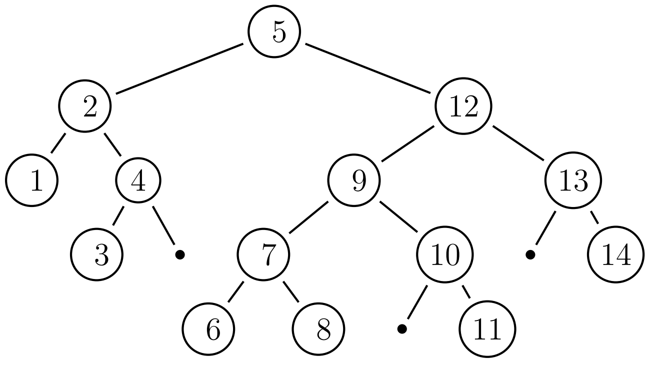
Given is an AVL tree. Perform the operation `insert(11)` on it. Balance the tree.



Solution

The operation `insert(11)` is shown in the figure - step by step.





Problem 8

For an ab-tree of height h (root node is at level zero) and n leaves, prove:

1. $2a^{h-1} \leq n \leq b^h$
2. $\lg_b(n) \leq h \leq \lg_a(\frac{n}{2}) + 1$

Solution

1. $2a^{h-1} \leq n \leq b^h$

   An ab-tree will have minimum number of nodes for a given height, when

   - the root node has only two children and
   - all the other nodes have the minimum number of children (i.e, `a` children)

   So, at height $= 1$ the number of nodes $= 2$, and at height $= 2$, the number of nodes is $= 2a$. Likewise, at height $= h$, the number of nodes is $2a^{h-1}$. Since this is the minimum possible, the first part of the inequality is satisfied.

   The tree will have maximum number of nodes at a given height, when all the nodes above that level has the maximum number of childres (i.e, `b` children). It is clear that the value is b^h for height h. Since this is the maximum possible value, the second part of the inequality also is satisfied.

2. $\lg_b(n) \leq h \leq \lg_a(\frac{n}{2}) + 1$

   The inequalities can be derived from the first part of the problem.

Problem 9

Show that the tree defined by the edges traversed in a BFS (starting at v_0) is a shortest paths tree rooted at v_0.

Solution

A complete mathematical proof based on induction is available on many texts and also available online. But that appears to be out of the scope of our course. The following proof give a more verbal treatment.

BFS lists all the vertices at level $k-1$ before those at level k. Therefore, it will place into the queue all vertices at level k before all those of level $k+1$ and therefore list the ones at k before those in level $k+1$. It is not possible for two vertices which are connected and have a difference of levels to be more than 1. ie, if a node is at level i and a connected node cannot be in level $i+2$. Because if they are connected, then that node should be added at level $i + 1$.

So BFS actually gives a shortest path tree starting at root.

- Every vertex has a path from/to root.
- The path length is equal to the level
- No path can skip a level hence the level will be always the minimum possible.

Hence the available path will be minimum path - hence the shortest paths tree.