Bilkent University
Computer Engineering Department

CS 202
Fundamental Structures of Computer Science
Section 1
Midterm Exam
Date: April 22, 2003, Tuesday
Duration: 120 minutes

Name of the Student	
ID of the Student	

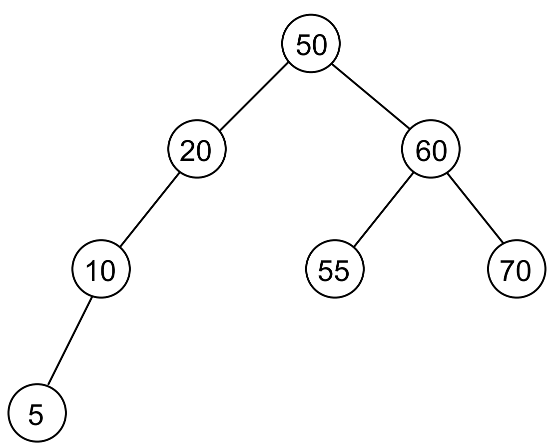
GRADE [ ]

- *Show your work and reasoning clearly!*
- *Write legibly!*
- *Write only to the space provided for each question!*
- *You can use the additional empty sheets as extra sheets to provide your answer or just as scratch sheets. If you use them as scratch sheets, just cross them or tear and throw them away at the end of the exam.*
- *There should be total of 14 questions. Check your exam paper.*

Good Luck!

1. Problem (4 points)

a) (*1 points*) Insert the following items into an initially empty *binary search tree*: 50, 20, 10, 60, 5, 70, 55. Draw the tree after all items are inserted (just show the final tree).
Answer:

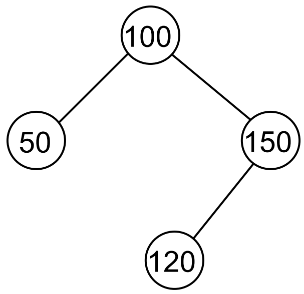


b) (*2 points*) How many nodes are there in a *complete binary tree* of height k? (Give minimum and maximum number of nodes)?
Answer:
Minimum = 2^k,
Maxiumum = 2^{k+1}-1

c) (*1 points*) How many nodes are there in a *full binary tree* of height k?
Answer:
2^{k+1}-1

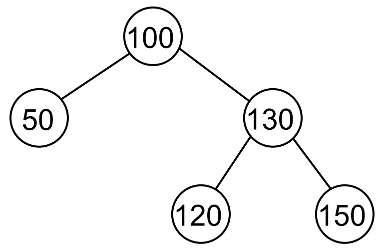
2. *Problem (9 points)*

The following AVL tree is given. Draw the new tree after every insertion (and rebalancing of the tree if necessary) of the following items: 130, 125, 20



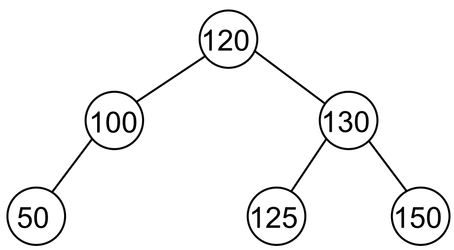
a) (*3 points*) After inserting 130

Answer:



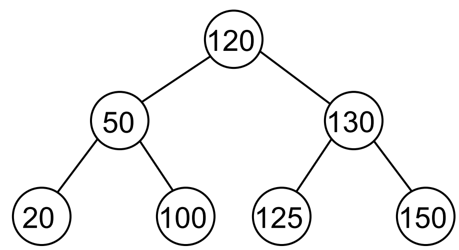
b) (*3 points*) After inserting 125

Answer:



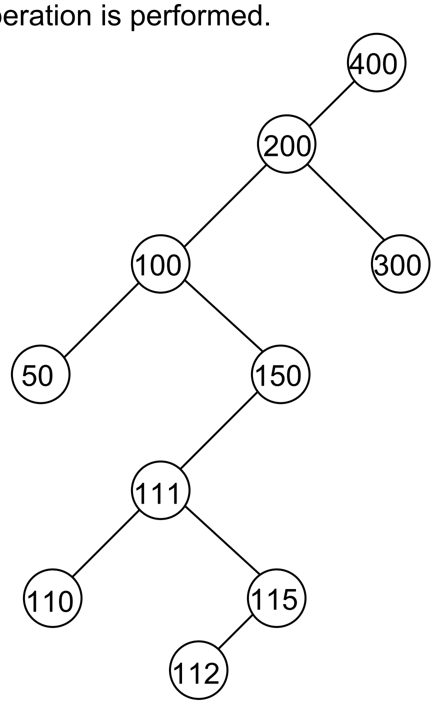
c) (*3 points*) After inserting 20

Answer:

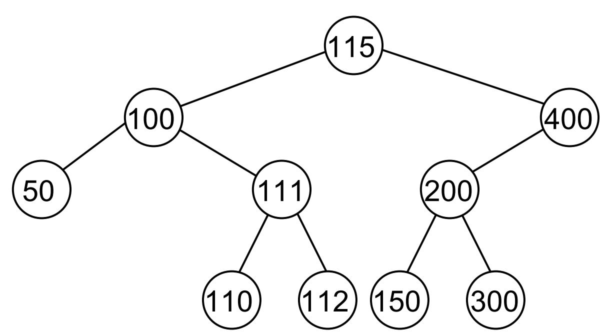


3. *Problem(5 points)*

The following splay tree is given. Draw the tree *after* item 115 is accessed and splaying operation is performed.



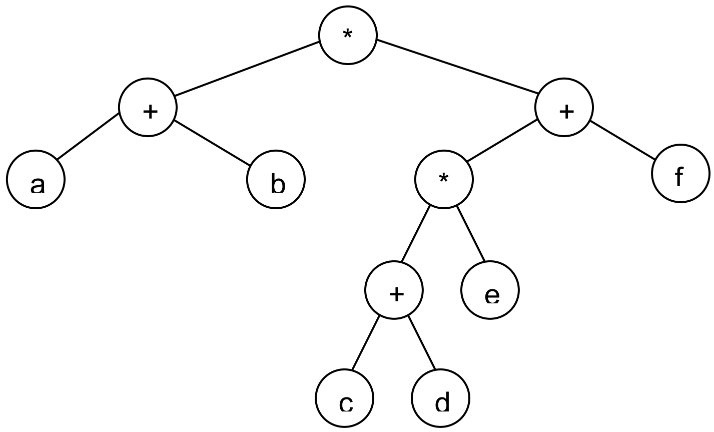
Answer:



4. Problem (6 points)

The following expression in *postfix* notation is given: a b + c d + e * f + *
(ab,c,d,e,f are operands, and +, * are operators).

a) (*3 points*) Draw the corresponding *expression tree!*
 Answer:

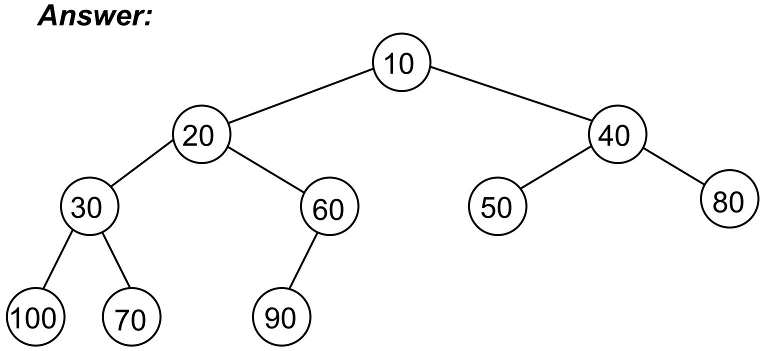


b) (*3 points*) Write down the same expression in *prefix* notation!
 Answer:
 * + a b + * + c d e f

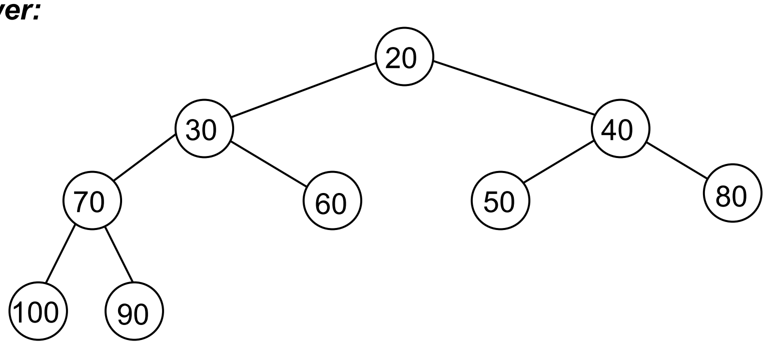
5. Problem (10 points)

The following array of items are given: 100, 90, 80, 70, 60, 50, 40, 30, 20, 10. In this array of items*, item 100* is located at array index 1 and *item 10* is located at array index 10. Answer the following questions.

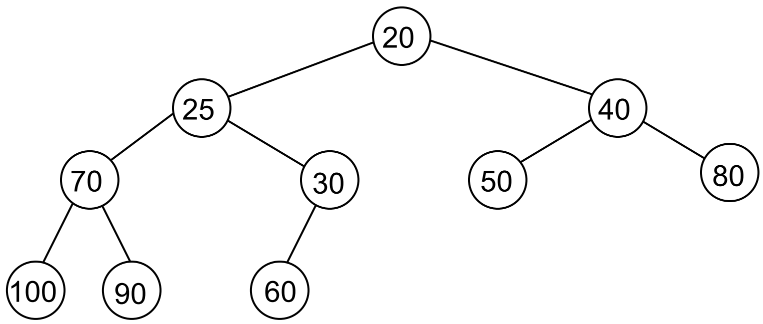
a) (*4 points*) Assume you apply the efficient (O(N) running time) *buildHeap* operation on this array to obtain a min-heap. Draw the resulting min-heap after this operation is performed (you can give it as a tree or as an array)
 Answer:



b) (*2 points*) Draw the heap again after you perform *deleteMin* operation.
 Answer:

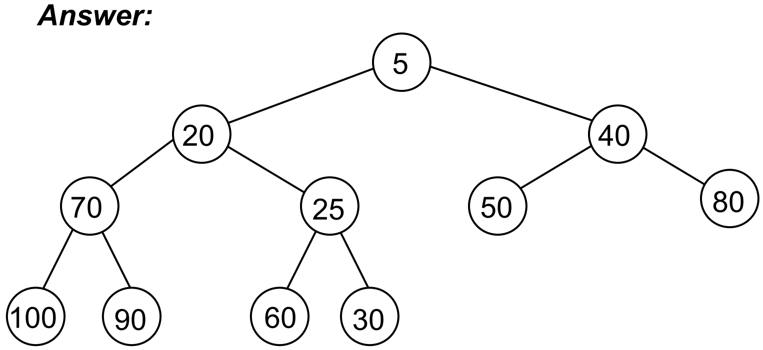


c) (*2 points*) Draw the heap after you *insert* item 25 into the heap.
 Answer:



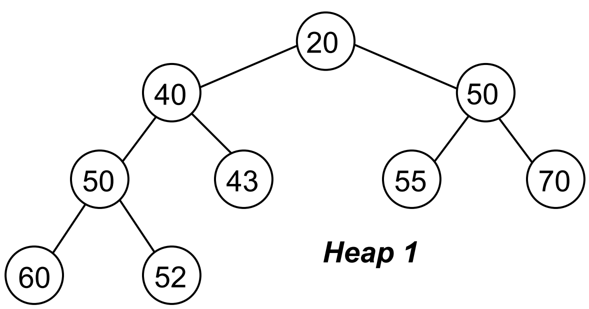
d) (*2 points*) Draw the heap after you insert item 5.

Answer:

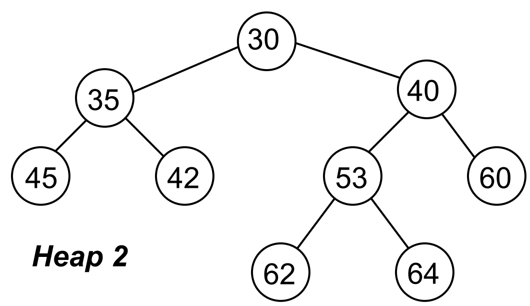


6. Problem (6 points)

Draw the resulting leftist heap after you merge the following leftist heaps.

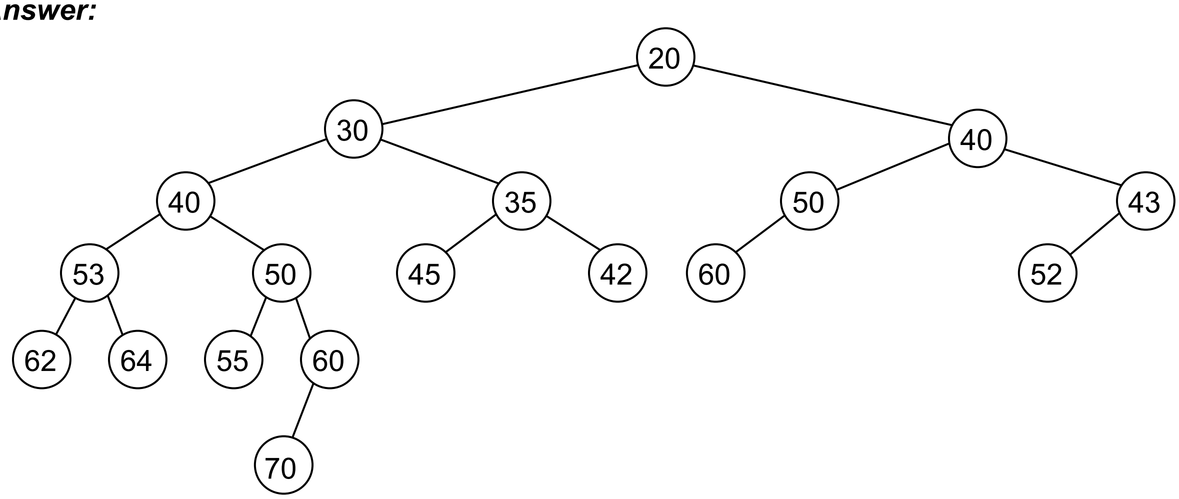


Heap 1

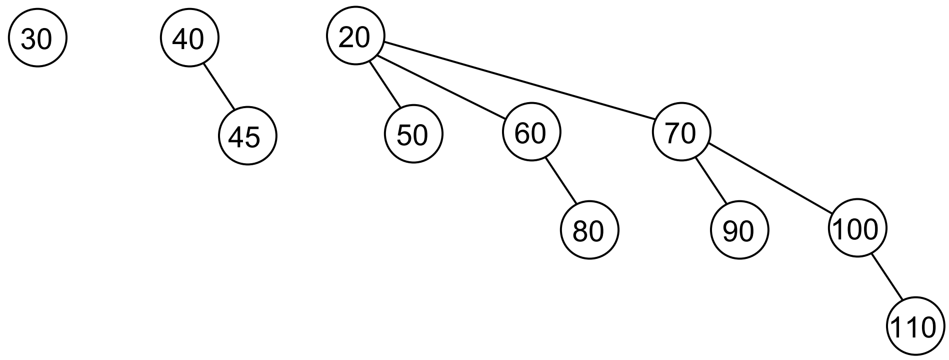


Heap 2

Answer:

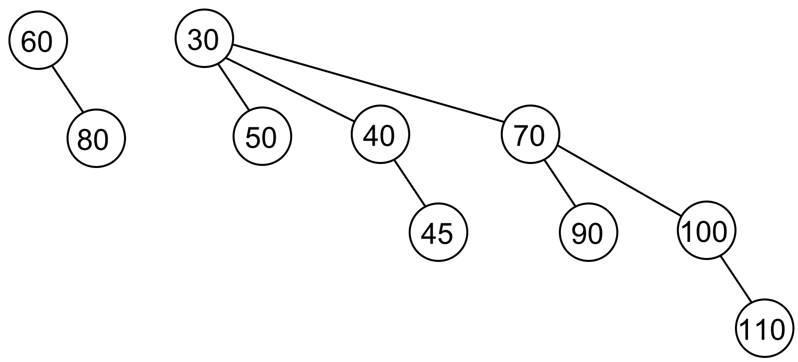


7. Problem (5 points)



Draw the resulting binomial heap after *deleteMin* operation is performed on the binomial heap shown above.

***Answer*:**



8. Problem (14 points)

We have a hash table of size 13. Double hashing technique is used to resolve collisions. The hash functions are:

$hash_1(x) = x \bmod 13$

$hash_2(x) = 7 - (x \bmod 7)$

Show the state of the hash table after the following items are inserted in the given order: 30, 23, 43, 45, 16, 25, 81. (Use the table given below for your answer).

0	
1	
2	
3	43
4	30
5	
6	45
7	
8	16
9	81
10	23
11	
12	25

9. Problem (8 points)

```
void
avl_insert_multiple(int array[], int N)
{
        AvlTree avl;
        int i;

        for (i=0; i<N; ++i)
                avl.insert(array[i]);
                // insert ith element of array
}

void
bst_insert_multiple(int array[], int N)
{
        BinarySearchTree bst;
        int i;

        for (i=0; i<N; ++i)
                bst.insert(array[i]);
// insert ith element of array
}
```

Two functions *avl_insert_multiple*() and *bst_insert_multiple*() are given above. Function *avl_insert_multiple* takes an *array of N items* and an *integer N* as input, and inserts all the N items in the array into an initially empty AVL tree. Function *bst_insert_multiple* takes an *array of N items* and an *integer N* as input, and inserts all the N items in the array into an initially empty binary search tree. Answer the following questions:

a) (*4 points*) What is *worst case* and *average case* running times of *avl_insert_multiple* function? Express your answers in Big-Oh notation depending on N. Your answers should be as tight as possible.
   1) (*2 points*) Worst case:
      O(NlogN)

   2) (*2 points*) Average case:
      O(NlogN)

b) (*4 points*) What is *worst case* and *average case* running times of *bst_insert_multiple* function? Express your answers in Big-Oh notation depending on N. Your answers should be as tight as possible.
   1) (*2 points*) Worst case:
      $O(N^2)$
   2) (*2 points*) Average case:
      O(NlogN)

10. Problem (6 points)

Solve the following recurrence relation: T(N) = 2T(T-1) + 1. (Give a formula for T(N). Don't express it using Big-Oh notation). T(0) = 1

Answer:

$$T(N) = 2T(N-1) + 1$$
$$= 4T(N-2) + 2 + 1$$
$$= 8T(N-3) + 4 + 2 + 1$$

$$= kT(N-k) + 2^{k-1} + 2^{k-2} + 2^{k-3} + \ldots + 2^0$$

ket k=N.

$T(N) = 2^N + 2^N-1 = 2^{N+1}-1$

$T(N) = 2^{N+1}-1$

11. Problem (9 points)

For each of the following sorting algorithms, express their worst-case, average-case and best-case running times using Big-Oh notation. Your answers should be as tight as possible. (Just fill in the table).

Answer:

	Worst Case	*Average Case*	*Best Case*
QuickSort	$O(N^2)$	O(NlogN)	O(NlogN)
MergeSort	O(NlogN)	O(NlogN)	O(NlogN)
Insertion Sort	$O(N^2)$	$O(N^2)$	O(N)

12. Problem (4 points)

```
// computes 2^x. x should be greater or equal to zero.
int power_two(int x)
{       int i;

        p = 1;
        for (i=1, i<=x; ++i)
                p = p * 2;
        return(p);
}

void power_random(int N)
{
        int x;

        for (i=0; i<=N; ++i)
        {
              x = random(0,N);
              // gives a random number
              // between 0 and N. (0,N incuded)
              power_two(x);
        }
}
```

What is the running time of the *power_random* function whose pseudo-code is provided above? Express your answer in Big-Oh notation depending on N. Your answer should be as tight as possible.

Answer:

$O(N^2)$

13. Problem (6 points)

For a given binary tree T, lets E(T) be defined as the <u>sum of the depts of all *leaves*</u> in T.

Lets assume T is a *compelete binary tree* with N nodes (N > 0). Express E(T) as a function of N (Show your work about how you have derived your formula)

Answer:

(on the right)

$$k = \lfloor \log N \rfloor$$

Number of leaves (x) at depth k (last level) are :

$$x = N - 2^k + 1$$

Number of leaves (y) at depth k - 1 (one level higher than last level) are :

$$y = 2^{k-1} - \left\lceil \frac{N - 2^k + 1}{2} \right\rceil$$

$$E(N,k) = (xk) + (y(k-1))$$

$$E(N) = k\left(N - 2^k + 1\right) + (k-1)\left(2^{k-1} - \left\lceil \frac{N - 2^k + 1}{2} \right\rceil\right)$$

where $k = \lfloor \log N \rfloor$

14. (8 points)

```
Class Node
{
  private:
          integer item;
          Node * left;
          Node * right;
          // some more data members
          // that you may find necessary
  friend class BinaryTree;
}

Class BinaryTree
{

public:
          int findAverage()
           { // find the average of all
              //items in the tree
                    return(findAvg(root));
          }
private:
          int findAvg(Node *t);
          Node *root;
}
```

Write a *recursive* private function *findAvg(Node *t)* using the class definitions given above. You can expand these definitions, but you should use the given data members and methods as they are.

The private function *findAvg(Node *t)* finds out and returns the *average* all of items stored in the subtree root at *t*.

Show all modifications (additions, etc) about the given class definitions.

Answer:

```
Class Node
{
  private:
          integer item;
          Node * left;
          Node * right;
          // some more data members that
          // you may find necessary
          int sum;   // some of the items in
                   //the subtree rooted at this node
          int nodes; // number of nodes in the
                    // subtree root at this node
  friend class BinaryTree;
}

Class BinaryTree
{

public:
          int findAverage()
           { // find the average of all
              // items in the tree
                    return(findAvg(root));
          }
private:
          int findAvg(Node *t);
          Node *root;
}
int BinaryTree::findAvg(Node *t)
{
          if (t==NULL) {
                    return(0);
          }
          else
          {
                    findAvg(t->left);
                    findAvg(t->right);

                    t->sum = t->item;
                    t->nodes = 1;
                    if (t->left)  {
                       t->sum += t->left->sum;
                       t->nodes += t->left->nodes;
                    }
                    if (t->right) {
                      t->sum += t->right->sum;
                      t->nodes += t->right->nodes;
                    }
                    return(t->sum / t->nodes);
          }
}
```