



**Instructions:**

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula/fact sheet. No calculators or other computing devices may be used.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 6 questions, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- **Note that failure to return this test, or to discuss its content with a student who has not taken it, is a violation of the Honor Code.**

**Do not start the test until instructed to do so!**

Name \_\_\_\_\_ **Solutions** \_\_\_\_\_

**Pledge:** On my honor, I have neither given nor received unauthorized aid on this examination.

\_\_\_\_\_

- 1) A given set of  $n$  distinct numbers may be sorted by first building a binary search tree containing those numbers (inserting the numbers one by one), and then printing the numbers during a traversal.

(a) [8 pts] What is the big-O worst-case running time for this sorting algorithm?

In the worst case, the BST will be a “stalk” (a linear list), and inserting to such a BST of  $k$  nodes has cost  $k$ , so building the tree will cost:

$$1 + 2 + \dots + n-1 \text{ which is } O(n^2).$$

Traversing the BST to print the nodes will have cost  $O(n)$ .

So the total cost is  $O(n^2)$ .

(b) [8 pts] What is the big-O best-case running time for this sorting algorithm?

In the best case, the BST will be balanced, and inserting to a balanced BST of  $k$  nodes has cost  $\log(k)$ , so building the tree will cost:

$$\log(1) + \log(2) + \dots + \log(n-1) = \log(n-1)! \text{ which is } O(n \log n).$$

Traversing the BST to print the nodes will have cost  $O(n)$ .

So the total cost is  $O(n \log n)$ .

(c) [6 pts] Which standard binary tree traversal should be used?

To print the node values in proper order, an inorder traversal must be used.

- 2) [10 pts] Consider the function  $f(n) = 4n^2 + 2^n$ . Show that  $f(n)$  is in  $O(2^n)$ . Be sure to clearly state the relevant constants  $c$  and  $n_0$  from the big-O definition.

We must find constants  $c > 0$  and  $n_0 > 0$  such that  $f(n) \leq c 2^n$  for all  $n > n_0$ . It's pretty obvious that if  $n$  is large enough then  $4n^2 \leq 2^n$ , so if  $n$  is large enough then we have

$$f(n) = 4n^2 + 2^n < 2^n + 2^n = 2 \cdot 2^n$$

The question is how large must  $n$  be to give us the desired inequality? Trial and error reveals that:

n	1	2	3	4	5	6	7	8
$4n^2$	4	16	36	64	100	144	196	256
$2^n$	2	4	8	16	32	64	128	256

So if  $n \geq 8$  we have that  $4n^2 \leq 2^n$ ; therefore we can take  $n_0 = 7$  and  $c = 2$ .

Other pairs of values will work as well; if we increase  $c$  then  $4n^2$  will be dominated by  $2^n$  at a smaller value of  $n$ .

- 3) [10 pts] A list of records holding 40 bytes of data must be stored, on a system where each pointer requires 4 bytes of storage. We consider two alternatives: a doubly-linked list and an array of fixed dimension  $D$ . Let  $N$  be the number of records actually stored, where  $N \leq D$ . Considering only total storage cost as important, how large must  $N$  be in order for the array to be the better choice? Show your analysis.

For the doubly linked list (DLL), each node will require 8 bytes for pointers and 40 bytes for data, for a total of 48 bytes per node. Given that the DLL will store  $N$  nodes, the total storage cost would be  $48N$ . (If you want to include the cost of a head pointer, that's OK. It doesn't change the final result significantly.)

For the array of dimension  $D$ , each cell will require 40 bytes, so the total storage cost would be  $40D$ .

The array will be the better choice (space-wise) precisely when:  $40D < 48N$ . A little algebra reveals that this will be true if

$$N < (40/48) D \text{ or } N < (5/6) D$$

so the array would be preferred if it was about 83% full.

- 4) (a) [5 pts] Explain why the Full Binary Tree Theorem implies that any full binary tree must have an odd number of nodes.

The FBT says that the number of leaves  $L$  in a FBT is one more than the number of internal nodes  $I$ . That is,  $L = I + 1$ . Now the total number of nodes  $N$  is then

$$N = L + I = 2I + 1$$

which is odd.

- (b) [12 pts] Consider a full binary tree in which all nodes store 128 bytes of data, internal nodes also store two 4-byte child pointers, and leaf nodes store no pointers. Assuming the tree has  $n = 2k + 1$  nodes, calculate the total storage, the overhead, and the overhead fraction (all as functions of  $k$ ).

The FBT will have  $k$  internal nodes and  $k+1$  leaves, by the FBT Theorem. Thus, the total storage cost would be:

$$T = k(128 + 2 \cdot 4) + (k+1)(128) = 264k + 128$$

The overhead is simply the cost of the pointer storage, which would be:

$$O = k(2 \cdot 4) = 8k$$

The overhead fraction  $OF$  is defined as the overhead  $O$  divided by the total storage cost  $T$ :

$$OF = (8k) / (264k + 128) = k / (33k + 16)$$

5) [21 pts] You must keep track of some data. Your options are:

- 1) a binary search tree of records
- 2) a linked-list of records in unsorted order (any order you like)
- 3) an array-based list of records maintained in sorted order

Suppose that  $2^{20}$  insertions must be made, and then  $2^4$  searches must be performed. For each option, use the big-**O** **time** complexity results of each data structure to determine the costs associated with that structure in this situation. Since you know exactly how many insertions and searches must be performed, the cost values should be numbers, not expressed in terms of  $n$ . Use your analysis, select the most appropriate of the alternatives.

**Note that space cost is not a consideration.**

Option	Cost of $10^6$ insertions	Cost of 10 searches	Total Cost
BST	$20 \cdot 10^6$	200	$2 \cdot 10^7$
unsorted linked list	$10^6$	$5 \cdot 10^6$	$6 \cdot 10^6$
sorted array	$19 \cdot 10^6$	200	$10^7$

First you have to decide whether to do a best/average/worst case analysis. I'll do an average case analysis:

Recall that on average:

BST insertion is  $O(\log n)$  so  $10^6$  insertions would cost about  $10^6 \cdot \log(10^6)$  or about  $20 \cdot 10^6$   
 BST search is  $O(\log n)$  so 10 searches would cost about  $10 \cdot \log(10^6)$  or about 200

That makes the total cost for the BST on the order of  $20 \cdot 10^6$  or  $2 \cdot 10^7$ .

ULL insertion is  $O(1)$  so  $10^6$  insertions would cost about  $10^6 \cdot 1$  or  $10^6$   
 ULL search is  $O(n/2)$  so 10 searches would cost about  $10 \cdot 10^6 / 2$  or  $5 \cdot 10^6$

That makes the total cost for the ULL on the order of  $6 \cdot 10^6$ .

SA insertion is  $O(n/2)$  so  $10^6$  insertions would cost about  $10^6 \cdot \log(10^6/2)$  or about  $19 \cdot 10^6$   
 SA search is  $O(\log n)$  so 10 searches would cost about  $10 \cdot \log(10^6)$  or about 200

That makes the total cost of the SA about  $19 \cdot 10^6$  or nearly  $10^7$ .

So the most efficient choice would be the unsorted linked list. (Of course this would change if the number of searches were not so small in relation to the number of data elements inserted.)

A worst-case analysis yields essentially the same conclusions. A best-case analysis doesn't really make much sense on a practical level since the best case scenarios are so unlikely.

6) [20 pts] Write a recursive function, `Search( )`, that takes as input a binary tree (not a BST) and a value `k`, and returns `true` if the value `k` appears in the tree and `false` otherwise. Your function must conform to the `BinNode` ADT given below and to the following interface:

```
bool Search(BinNode* rt, const int K) {  
    if (rt == NULL) return false;  
    if (rt->getValue() == K) return true;  
    if (Search(rt->leftChild()) return true;  
    if (Search(rt->rightChild()) return true;  
    return false;  
}
```

OR if you prefer a “one-liner”:

```
bool Search(BinNode* rt, const int K) {  
    return ( (rt == NULL)  
            || (rt->getValue() == K)  
            || (Search(rt->leftChild())  
            || (Search(rt->rightChild()) )  
            )  
}
```

---

```
class BinNode {  
private:  
    int Element;  
    BinNode* Left, Right;  
    static BinNode* FreeList;  
public:  
    BinNode() {Left = Right = NULL;};  
    BinNode(int val, BinNode* l = NULL, BinNode* r = NULL)  
        {Element = val; Left = l; Right = r;}  
    ~BinNode();  
    BinNode* leftChild() const {return Left;}  
    BinNode* rightChild() const {return Right;}  
    int getValue() {return Element;}  
    void setValue(int val) {Element = val;}  
    bool isLeaf() const {return (Left == NULL && Right == NULL);}  
    void* operator new(size_t);  
    void* operator delete(void* );  
};
```