

**Quiz 3 – 20 points possible**  
**4 questions on 5 pages**

*(You should spend no more than 2 minutes on this page)*

1. (6pts) Give the Big-Oh running times of the following operations:

- a) Enqueue into a heap
  
- b) Dequeue from a heap
  
- c) Performing treesort (**average case**)
  
- d) Performing heapsort
  
- e) Remove from a binary search tree (**worst case**)
  
- f) Insert into a binary search tree (**average case**)

*(You should spend no more than 5 minutes on this page)*

2. (4 pts) When you use a word processors spell checker, it compares words in your document with words in a dictionary. You can add new words to the dictionary as necessary. Thus the dictionary needs frequent retrievals and occasional insertions. What ADT would be most appropriate as a spell checkers dictionary? Be sure to discuss size requirements, runtimes, and anything else appropriate. Give as much information as necessary because you will be mostly graded on how you arrived at your answer and not what your final ADT choice is.

Continued on next page

---

*(You should spend no more than 5 minutes on this page)*

3. (6 pts) Given the following numbers: 1, 2, 3, 4, 5, 6, and 7 draw the following:

a) A binary search tree of maximum height

b) A binary search tree of minimum height

c) A min heap (draw the heap as a tree representation and not an array)

Continued on next page

---

(You should spend no more than 5 minutes on this page)

4. (4 pts) Write the function search for a binary search tree. **Your function must be recursive.** Search will take as a parameter the item to look for in the tree and will return a pointer to that node or null if the item does not exist. Be sure to use good style throughout. Assume that the following functions and classes have been defined for you:

```
class Tree {
private:
    Node* root;
    Node* search (Node*, itemtype);
public:
    Node* search (itemtype);
};

Node* Tree::search ( itemtype key ) {
    return search (root, key);
}
```

```
class Node {
friend class Tree;
private:
    itemtype item;
    Node* left;
    Node* right;
};
```

```
Node* Tree::search(Node* current, itemtype key) {
```

**Extra Credit (2pts)**

Explain why a tree with  $N$  nodes has exactly  $N-1$  edges.

DONE =>

---