# CpSc 2120: Algorithms and Data Structures

**Instructor:** Dr. Brian Dean — Fall 2019
**Webpage:** `http://www.cs.clemson.edu/~bcdean/` — MW 2:30-3:45
**Handout 10:** Quiz 1 (In Class). 30 possible points. — Daniel 313

**1. True/False (1 point each).** Please circle T (true) or F (false):

T F A skip list made of $n$ elements has height $O(n^2)$.

T F Take a binary tree representing an arbitrary sequence (the variant of binary search tree we studied in lab 5, which we interact with using rank-based access instead of value-based access). If we pick some node $x$ in the tree and repeatedly rotate $x$ with its parent until $x$ becomes the root, this can potentially change what is printed out during an in-order traversal of the tree.

T F If an operation runs in $O(n)$ time and an algorithm calls this operation $n$ times, then the algorithm runs in $\Omega(n^2)$ time.

T F If we insert the integers $1, 2, \ldots, n$ into an AVL tree, this takes $O(n \log n)$ time.

T F If we insert the integers $1, 2, \ldots, n$ into a splay tree, this takes $\Omega(n \log n)$ time.

T F If we `push` $n$ elements successively onto a stack and then call `pop` $n$ times, the elements will be output in reverse order.

T F The worst-case running time of randomized quicksort applied to an already-sorted array is $\Theta(n^2)$.

T F If a C++ class has a private integer member variable $x$, then a public function in the class can return a pointer to $x$, allowing $x$ to be changed by a function that is not a class member.

T F Inserting $n$ elements into a hash table of size $\sqrt{n}$ (using chaining to resolve collisions, and not checking if duplicates are present) can be done in $O(n)$ time.

T F A in-order traversal of an $n$-element binary search tree always takes $O(n)$ time, even if the tree is not balanced.

**2. Heap (4 points).** Suppose you insert the numbers $5, 4, 3, 2, 1$ in that order into a standard "min" binary heap (a heap designed for easy extraction of the minimum element) that is initially empty. Please write down the contents of the array representing the heap afterward.

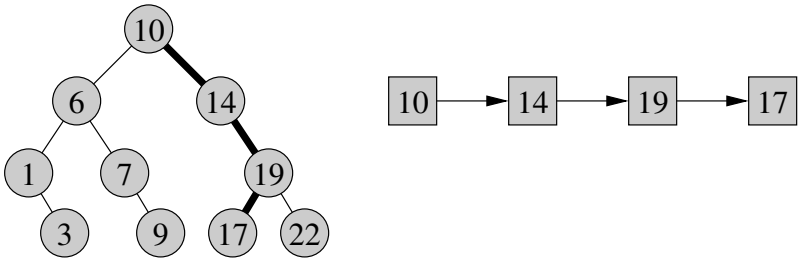


Figure 1: Returning the path from the root down to the element of key $x = 17$ as a linked list.

**3. TreePath (4 points).** Let us define nodes for a linked list and for a binary search tree in the following standard way:

```
struct ListNode {
  int key;
  ListNode *next;
  ListNode(int k, ListNode *n) { key = k; next = n; }
};

struct TreeNode {
  int key;
  TreeNode *left, *right;
};
```

Please complete the following function that takes a pointer to the root of a balanced binary search tree and an integer $x$ returns a pointer to the head node of a newly-allocated linked list representing the path in the tree from the root down to $x$. The figure above shows an example. If $x$ does not exist in the tree, the function should return NULL.

```
ListNode *getpath(TreeNode *root, int x)
{
```

**4. Similar Elements (4 points).** Consider binary search trees defined using the following standard node structure:

```
struct Node {
  int key;
  Node *left, *right;
};
```

You are given pointers to the roots of two binary search trees $T_1$ and $T_2$, each containing $n/2$ elements. Please describe, preferably in English rather than code, an efficient algorithm for determining whether there is some element $x$ in $T_1$ and some element $y$ in $T_2$ such that $x$ and $y$ differ in value by at most 7 (i.e., $|x - y| \leq 7$). Give the running time of your algorithm – the faster the asymptotic running time of your algorithm, the more points you will receive.

**5. Black Boxes (4 points).** You have just completed an implementation of three sorting algorithms: merge sort, insertion sort, and bubble sort. Each one takes a file of $n$ numbers as input and writes out a file of the $n$ numbers in sorted order as output. Unfortunately, after compiling your three programs, you have forgotten which one is which!

(a) Describe how you can figure out, via some manner of computational testing, which of the three programs is *merge sort*.

(b) You have identified which of the three programs is merge sort. Please now describe how you can figure out, via some manner of computational testing, which of the two remaining programs is *bubble sort*.

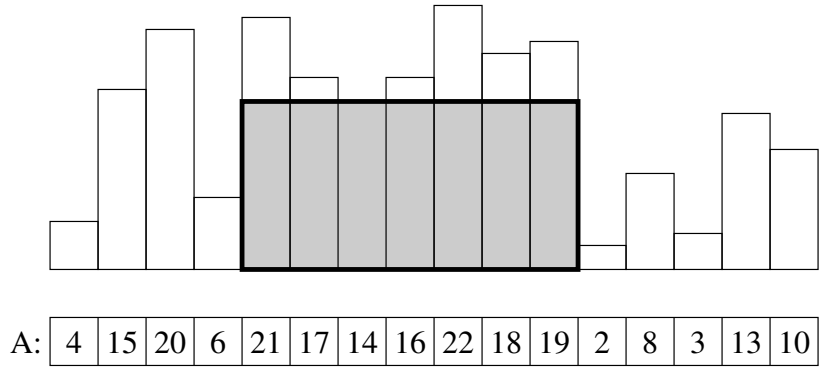


Figure 2: A rectangular building contained in height by a bar graph.

**6. Building (4 points).** You want to construct a new building along an $n$-meter long street. However, the local building codes for this street are quite bizarre. They come in the form of an array $A[0 \ldots n-1]$ of distinct integers, where $A[0]$ is the maximum height of any building whose facade partially lies within the first meter of the street, $A[1]$ is the maximum height of any building whose facade partially lies within the second meter of the street, and so on. The figure above shows how we can think of this visually as a "bar graph" that constrains the maximum height of the facade of a building at various points along the street. You would like to determine the maximum area of the facade of a rectangular building you can construct subject to these constraints. That is, what is the largest rectangle you can fit inside the bar graph described by the array $A$. In the example above, the largest area is formed by rectangle with width 7 and height 14.

Please describe an efficient algorithm for determining this maximum possible area and give its running time.

This page may be used for scratch work.

This page may be used for scratch work.