CS_MATH395_S17_A2_Brewster_Brandon

Problem 1)

Let A = a1, ..., an and B = b1, ..., bm be two sets of numbers. Consider the problem of finding their intersection, i.e., the set C of all the numbers that are in both set A and set B.
a) Design a brute-force algorithm for solving this problem.

```
vector<int> intersection( vector<int> a, vector<int> b ) {
    vector<int> intersection;
    for(int i=0; i<a.size(); i++) {
        for(int j=0; j<b.size(); j++) {
            if(a[i] == b[j] && !memberOf(intersection, a[i])) {
                intersection.push_back(a[i]);
            }
        }
    }
    return intersection;
}
```

b) Determine the efficiency class of this brute-force algorithm

$O(nm)$

c) Design a presorting-based algorithm for solving this problem

```
vector<int> intersection( vector<int> a, vector<int> b ) {
    vector<int> intersection;
    for(int i=0; i<a.size(); i++) {
        int bs = binarySearch(b, a[i], 0, b.size());
        if(bs && !memberOf(intersection, a[i])) {
            intersection.push_back(a[i]);
        }
    }
    return intersection;
}
```

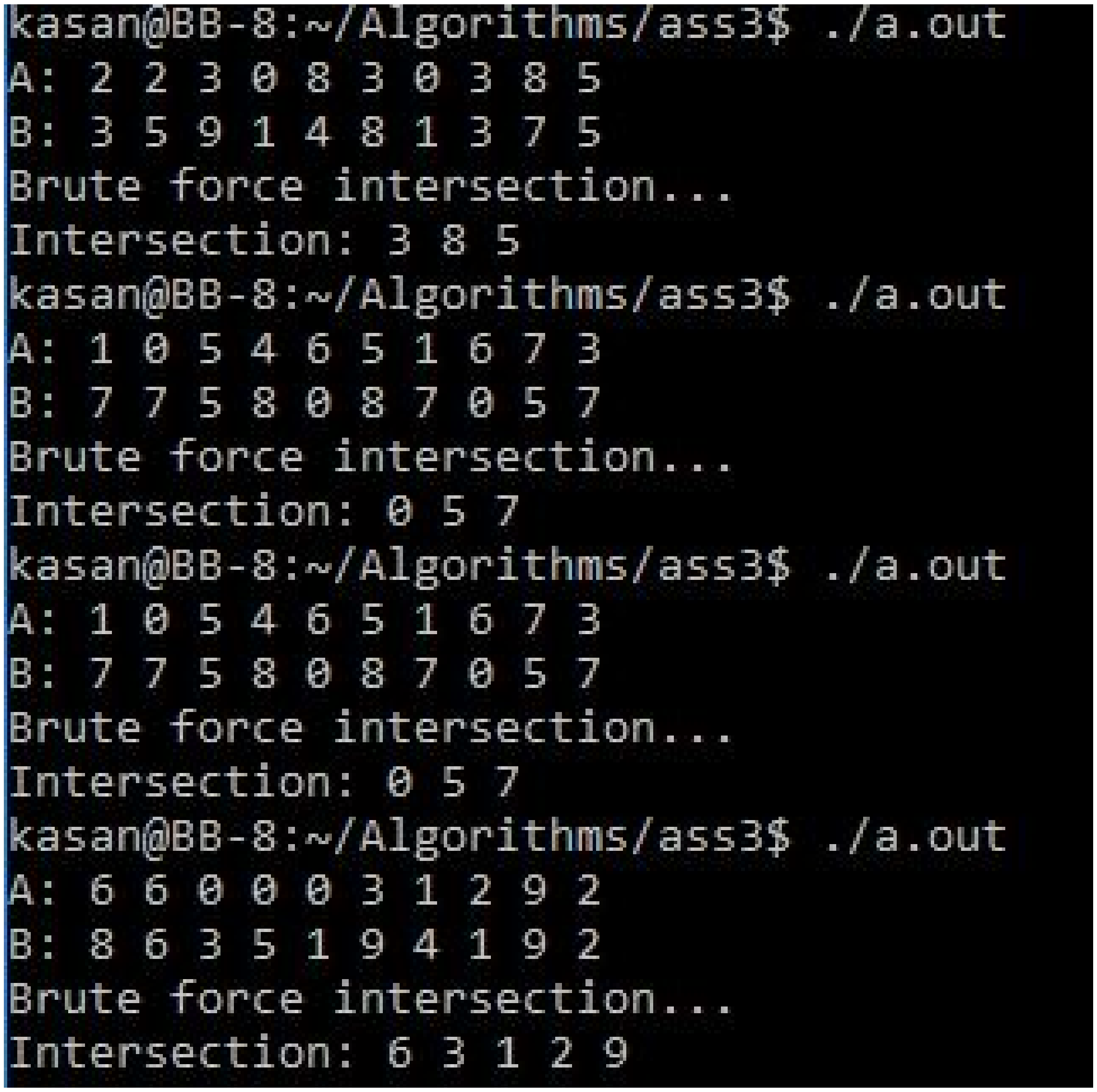
d) Determine the efficiency class of this presorting-based algorithm

$mO(\log n)$

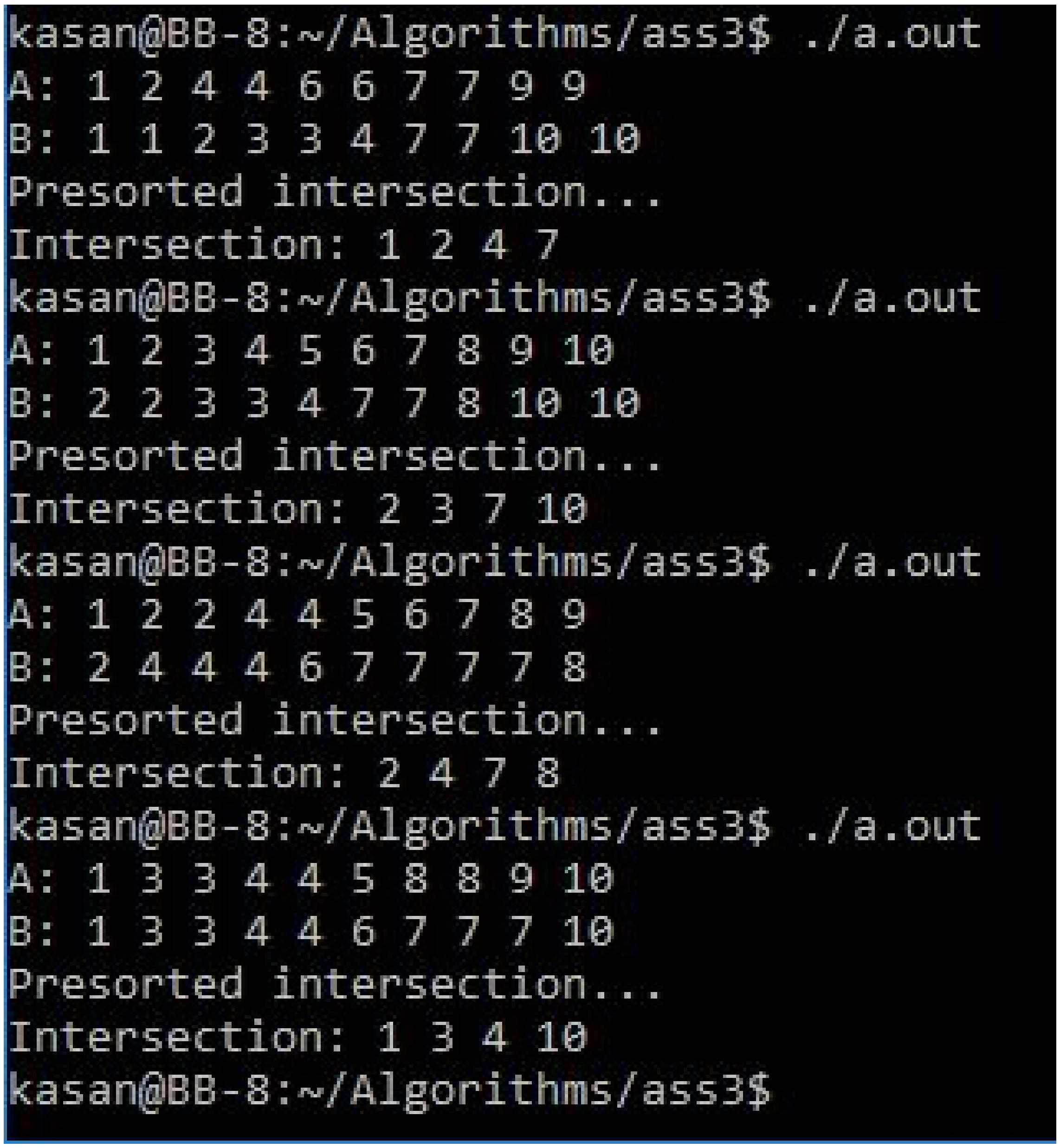
e) Show how much improvement the second algorithm has achieved (if any) over the first one in terms of time efficiency

The time efficiency of the search is reduced from $O(n)$ to $O(\log n)$

f) Implement both the algorithms (using C/C++). Run the programs on same set of data and record and report the system-times for them. Compare the results with your answer of 1.e problem above



```
kasan@BB-8:~/Algorithms/ass3$ ./a.out
A: 2 2 3 0 8 3 0 3 8 5
B: 3 5 9 1 4 8 1 3 7 5
Brute force intersection...
Intersection: 3 8 5
kasan@BB-8:~/Algorithms/ass3$ ./a.out
A: 1 0 5 4 6 5 1 6 7 3
B: 7 7 5 8 0 8 7 0 5 7
Brute force intersection...
Intersection: 0 5 7
kasan@BB-8:~/Algorithms/ass3$ ./a.out
A: 1 0 5 4 6 5 1 6 7 3
B: 7 7 5 8 0 8 7 0 5 7
Brute force intersection...
Intersection: 0 5 7
kasan@BB-8:~/Algorithms/ass3$ ./a.out
A: 6 6 0 0 0 3 1 2 9 2
B: 8 6 3 5 1 9 4 1 9 2
Brute force intersection...
Intersection: 6 3 1 2 9
```



```
kasan@BB-8:~/Algorithms/ass3$ ./a.out
A: 1 2 4 4 6 6 7 7 9 9
B: 1 1 2 3 3 4 7 7 10 10
Presorted intersection...
Intersection: 1 2 4 7
kasan@BB-8:~/Algorithms/ass3$ ./a.out
A: 1 2 3 4 5 6 7 8 9 10
B: 2 2 3 3 4 7 7 8 10 10
Presorted intersection...
Intersection: 2 3 7 10
kasan@BB-8:~/Algorithms/ass3$ ./a.out
A: 1 2 2 4 4 5 6 7 8 9
B: 2 4 4 4 6 7 7 7 7 8
Presorted intersection...
Intersection: 2 4 7 8
kasan@BB-8:~/Algorithms/ass3$ ./a.out
A: 1 3 3 4 4 5 8 8 9 10
B: 1 3 3 4 4 6 7 7 7 10
Presorted intersection...
Intersection: 1 3 4 10
kasan@BB-8:~/Algorithms/ass3$
```

Problem 2)

For n = 1, 2, 3, 4, and 5, draw all the binary trees with n nodes that satisfy the balance requirement of AVL trees.

n=1

0

n=2

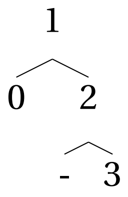
$$\begin{tikzcd}[column sep=small] & 1 \arrow[dl, dash] \arrow[dr, dash] & \\ 0 & & - \end{tikzcd}$$

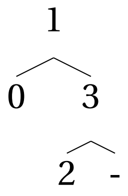
$$\begin{tikzcd}[column sep=small] & 0 \arrow[dl, dash] \arrow[dr, dash] & \\ - & & 1 \end{tikzcd}$$

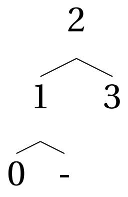
n=3

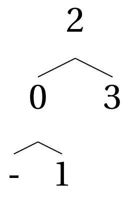
$$\begin{tikzcd}[column sep=small] & 1 \arrow[dl, dash] \arrow[dr, dash] & \\ 0 & & 2 \end{tikzcd}$$

n=4

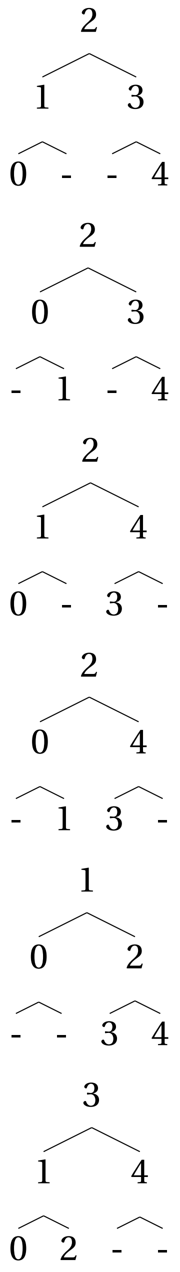






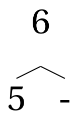


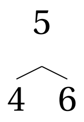
n=5

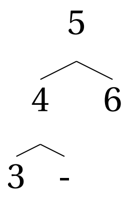


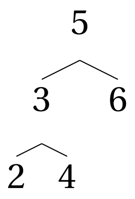
Problem 3)

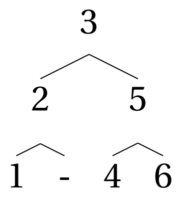
Consider the list [6, 5, 4, 3, 2, 1]. Construct an AVL tree by inserting the elements of this list successively, starting with the empty tree.











Problem 4

a) Design an efficient algorithm for finding and deleting an element with the smallest value in a heap and determine its time efficiency.

```
// We can assume that the smallest number will be in a leaf position
// This means that it will always be in the second half of the list
// by scanning the second half of the list sequentially we can delete
// the value in Θ(log n)
int deleteMin( vector<int> heap )
{
    int midPoint = heap.size()/2 + 1;
    int min = midpoint;
    for(int i=0; i<heap.size(); i++) {
        if(heap[i] < heap[min])
            min = i;
    }
    heap.remove(min);
    heap.shiftToFill;
}
```

b) Design an efficient algorithm for finding and deleting an element with a given value v in a heap H and determine its time efficiency.

```
// In this case we have to search the whole heap in O(n) time.
// deleting the element requires a complete restructure of the heap.
// This is done in Θ(log n) time
int deleteKey( vector<int> heap, int key)
{
    for(int i=0; i<heap.size(); i++) {
        if(heap[i] = key)
            heap.remove(i);
    }
    heap.shiftToFill();
}
```

problem 5)

Implement Horspool's algorithm, the Boyer-Moore algorithm, and the brute-force algorithm of Section 3.2 (using C/C++) and run an experiment to compare their efficiencies for matching random natural-language patterns in natural-language texts.

Problem 6)

Explain how to use hashing to check whether all elements of a list are distinct. What is the time efficiency of this application? Compare its efficiency with that of the bruteforce algorithm (Section 2.3) and of the presorting-based algorithm (Section 6.1).

In order to check uniqueness, initialize a hash table that is the same size as the list. Populate the hash table by running each element of the list through the hash function. Any values that are the same will cause a collision in the hash table. Checking for collisions is not enough however, as there are cases where items with different values can generate the same hash value. Compare the two items when a collision is found. This brings the overall time complexity down to $O(n)$ where brute force was $O(n^2)$ and presorting was $O(n \log n)$

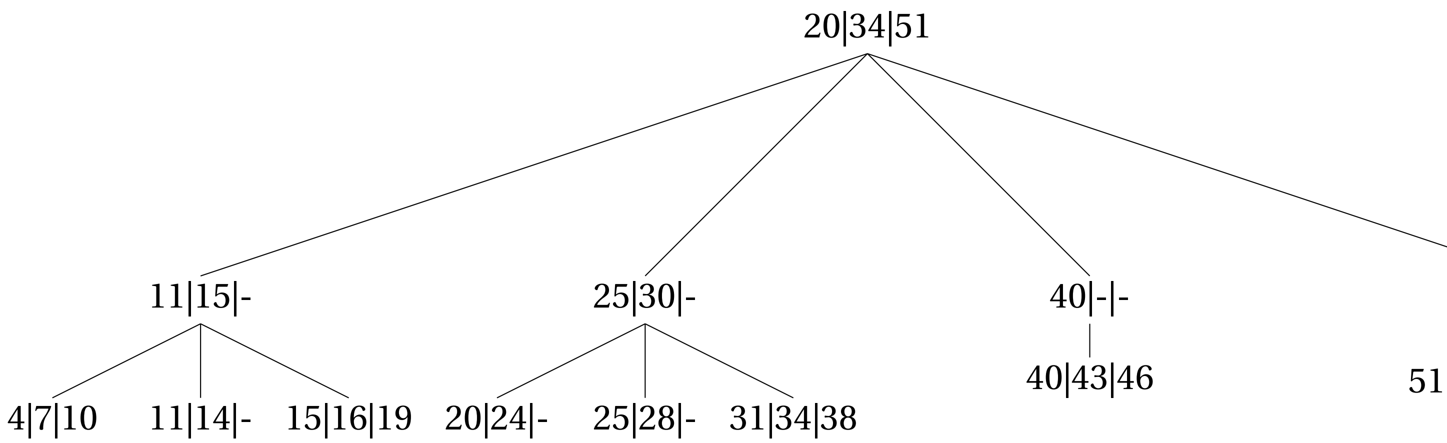
Problem 7)

Fill in the following table with the average-case and worst-case efficiency classes for the five implementations of the ADT dictionary:

	unordered		ordered array		binary st		balanced st		hashing	
	avg	worst	avg	worst	avg	worst	avg	worst	avg	worst
search	$\Theta(n)$	$O(n)$	$\Theta(\log n)$	$O(\log n)$	$\Theta(\log n)$		$\Theta(1)$		$\Theta(1)$	
insert	$\Theta(1)$	$O(n)$	$\Theta(n)$	$O(n)$	$\Theta(\log n)$		$\Theta(1)$		$\Theta(1)$	
delete	$\Theta(1)$	$O(n)$	$\Theta(n)$	$O(n)$	$\Theta(\log n)$		$\Theta(1)$		$\Theta(1)$	

Problem 8)

Draw the B-tree obtained after inserting 30 and then 31 in the B-tree in the Figure below. Assume that a leaf cannot contain more than three items.



Problem 6)

Write pseudocode for a divide-and-conquer algorithm for finding values of both the largest and smallest elements in an array of n numbers

pseudocode:

```
int min( vector<int> list )
{
   if( list.size() == 1 ) {
      return list[0];
   }
   vector<int> left;
   vector<int> right;
   for( int i=0; i<list.size(); i++ ) {
      if( i%2 == 0 )
         left.push_back(list[i]);
      else
         right.push_back(list[i]);
   }
   int leftmin = min( left );
   int rightmin = min( right );
   if( leftmin > rightmin )
      return rightmin;
   else
      return leftmin;
}
```

a) Setup and solve (for $n = 2^k$) a recurrence relation for the number of key comparisons made by you algorithm.

$$M(n) = M(n-2) + 1 \text{ for } n > 2$$
$$\text{where } M(2) = 1, M(1) = 0.$$

b)] How does this algorithm compare with the brute-force algorithm for this problem?

The brute force algorithm will be the same for this, as there will still be the same number of comparisons made. Every value must be checked.