

COP 3530
Data Structures

Midsemester Exam

Name: _____

October 7, 1999

This exam has 4 questions. Each question starts on a new page. Each page is worth 50 points. Please answer each question on its page. You may write on the back of a page.

1. This question refers to the function `canSum` defined below. `canSum` returns `true` if there are two integers in the array `A` that sum to exactly `K`. Ignore all syntax errors.

```
bool canSum( const vector<int> & a, int k )
{
    for( int i = 0; i < a.size( ); i++ )
        for( int j = i + 1; j < a.size( ); j++ )
            if( a[ i ] + a[ j ] == k )
                return true;

    return false;
}
```

- (a) What is the Big-Oh running time of this function?
- (b) Suppose it takes 64 seconds to run `canSum` on an array of 100000 elements on a computer *A*. Computer *B* is 64 times faster than computer *A*. How large a problem can be solved on computer *B* in 16 seconds?
- (c) Consider the following alternative to the algorithm above:

```
bool canSum( const vector<int> & a, int k )
{
    set<int> s;
    for( int i = 0; i < a.size( ); i++ )
        s.insert( a[ i ] );

    for( int i = 0; i < a.size( ); i++ )
        if( s.find( k - a[ i ] ) != s.end( ) )
            return true;

    return false;
}
```

Assume that calls to `s.find` and `s.insert` take $O(\log N)$ time. What is the running time of the alternative algorithm?

2. Write a function template that takes a vector of objects and returns the object that appears most often in the vector. In cases of ties, you may return any of the tied objects. You may assume that the objects have a total order.

For instance, if the input contains 4, 3, 5, 6, 3, 2, 3, 4, then your function will return 3. Use a `map` to store counts for each object; then iterate through the map, to determine the most-frequently occurring object.

Complete the implementation of `mostFrequent` that is started below.

```
template <class Comparable>
Comparable mostFrequent( const vector<Comparable> & v )
{
```

3. Function `splitList` accepts an STL list `theList` and constructs two STL lists `evens` and `odds`. After `splitList` returns, `evens` contains the items in `theList` that are in even positions (the first item is an odd position) in the original order, while `odds` contains the items in `theList` that are in odd positions, in reverse order.

For instance, if `theList` contains 3, 4, 1, 7, 5, then when the call returns, `evens` contains 4, 7, and `odds` contains 5, 1, 3. Complete the implementation of `splitList` that is started below.

```
template <class Object>
void splitList( const list<Object> & theList, list<Object> & evens,
               list<Object> & odds )
{
    list<Object>::const_iterator itr;    // to traverse theList
```

4. Multiple choice (10 points each)

- (a) Which of the following is a linear-time process on average?
 - i. insertion sort
 - ii. merge sort
 - iii. quick sort
 - iv. quick select
 - v. none of the above
- (b) Which of the following uses linear extra space on average?
 - i. insertion sort
 - ii. merge sort
 - iii. quick sort
 - iv. quick select
 - v. none of the above
- (c) What is the average-case running time of quicksort?
 - i. $O(\log N)$
 - ii. $O(N)$
 - iii. $O(N^2)$
 - iv. $O(N \log N)$
 - v. none of the above
- (d) For quicksort, what do i and j do when they see keys equal to the pivot?
 - i. i stops, j stops
 - ii. i stops, j goes
 - iii. i goes, j stops
 - iv. i goes, j goes
 - v. i and j alternate between stopping and going
- (e) In median-of-three partitioning, to what position does the pivot initially get moved to?
 - i. 0
 - ii. low
 - iii. $(\text{low} + \text{high}) / 2$
 - iv. $\text{high} - 1$
 - v. high