CSci 115, Spring 2015, Midterm 2

Name (please print): _____ ID #: _____

- This is a 75-minute closed-book exam. There are **six** questions worth a total of 100 points. Budget your time so you get to each question.
- Please read **all** of the instructions carefully before beginning to work on a problem, and if anything is not completely clear, please raise your hand and ask.
- All code is C++.

1.	_____	5.	_____
2.	_____	6.	_____
3.	_____		
4.	_____	Total	_____

(for grader use only)

Question 1 (22 points). Complete the following definitions of `quicksort` and `partition`:

```
void quicksort(int A[], int i, int j) {
  int pivotindex = (i+j)/2;
  swap(A, pivotindex, _______);
  int k = partition(A, _______, _______, _______);
  swap(A, _______, _______);                    // Place pivot
  ___________________________________________;  // Sort left part
  ___________________________________________;  // Sort right part
}


int partition(int A[], int left, int right, int pivot) {









}
```

Question 2 (18 = 10 + 8 points). Invariants

(a) Here is the code for Insertion Sort, with two locations in the code labeled `/*1*/` and `/*2*/`:

```
void inssort(int A[], int n) {
   for (int i = 1; i < n; i++)
     /* 1 */
     for (int j = i; j > 0; j--)
        /* 2 */
        if (A[j] < A[j-1])
           swap(A, j, j-1);
        else
           break;
}
```

List the invariant(s) below that are needed for the proof of correctness of this algorithm. You may use abbreviations such as "a ≤ A[i..j]" and "A[i..j] ≤ A[r..s]" in your answers.

Invariant(s) true at /* 1 */ Invariant(s) true at /* 2 */

(b) Here is an implementation of binary search, with a location in the code labeled /*1*/:

```
int binary(int A[], int n, int k) { // assume A[0..n-1] sorted
   int left = 0, right = n-1;
   while (left <= right) {
      /* 1 */
      int mid = (left + right)/2;
      if (k < A[mid]) right = mid - 1;
      else if (k > A[mid]) left = mid + 1;
      else return mid;
   }
   return -1;
}
```

Complete the statement (inv) below so that it correctly represents the property of the return value, m:

(inv) if m == `binary(A,n,k)`, then _____

Give the invariant for /*1*/ that is needed for the proof of (inv). Remember, it has to be true every time /*1*/ is reached and strong enough to imply (inv) when the function returns.

Invariant(s) true at /* 1 */

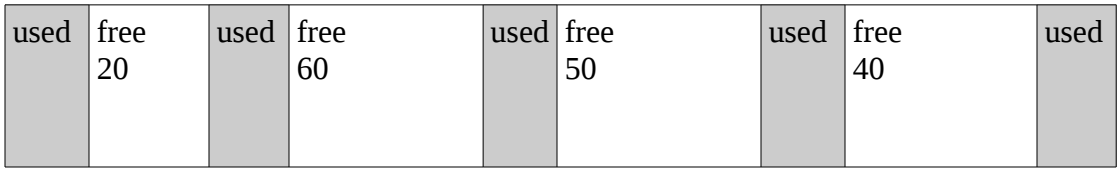
Question 3 (12 points). Hash update. Suppose we are given the following declarations for implementing closed hashing:

```
class KVpair {        // Key-Value pairs
   public:
      string key;
      int val;
}

const int M = 1001;   // hash array size
KVpair HA[M];         // hash array

// hash function, returns a value in the range 0..M-1
int h(string s);

// probe function, can return arbitrary positive number (i.e., not 0)
int p(int k, int i);
```

Complete the definition of the function `update` below, which **inserts** a key and value into the hash array `HA` if the key is not already there, and **overwrites** the existing value for a key that is already there (you can assume that the update will succeed; i.e., that probing will eventually find an empty slot):

```
void update(const string key; const int val) {









}
```

Question 4 (12 points). Memory Management. At a certain point in the execution of a program, the memory management system has free blocks of sizes 20, 60, 50, and 40, in that order:



New requests come in for a block of size 30, followed by a block of size 15. Assuming that there is no space overhead involved in storing free blocks, and unused portions of blocks are inserted back into the free list at the same spot, what are the sizes of the four blocks that remain on the free list (in order) after handling these two requests, if the policy used is

(a) first fit? _____ , _____ , _____ , _____

(b) circular fit? _____ , _____ , _____ , _____

(c) best fit? _____ , _____ , _____ , _____

(d) worst fit? _____ , _____ , _____ , _____

Question 5 (15 points). Sorting characteristics. Fill out the table below by doing the following for each column of the table (i.e., for each sorting algorithm):

- Circle either n, n log n, or n^2 in the cells for "Best case", "Average case", and "Worst case" to indicate the time complexity of the algorithm in these cases.
- Write the numbers 1, 2, 3, and 4 into the four subcells of the "Best array" cell in the order that would require the algorithm to do the **least** amount of work sorting this 4-element array, where by "work" I mean the total number of comparisons and swaps.
- Write the numbers 1, 2, 3, and 4 into the four subcells in the "Worst array" cell in the order that would require the algorithm to do the **most** amount of work sorting this 4-element array.
- If there is more than one array that produces the same best/worst case behavior, then you may choose any of the available candidates.

	Insertion	Bubble	Selection	Merge	Quick*
Best case	n n log n n^2	n n log n n^2	n n log n n^2	n n log n n^2	n n log n n^2
Average case	n n log n n^2	n n log n n^2	n n log n n^2	n n log n n^2	n n log n n^2
Worst case	n n log n n^2	n n log n n^2	n n log n n^2	n n log n n^2	n n log n n^2
Best array					
Worst array					

*__Note__: QuickSort is the one given on the first page of the exam, where `pivotindex = (i+j)/2`.

Question 6 (21 = 7 x 3 points). Short Answer. Be brief, but cover the main points.

(a) Give an example of a technique that can be used to improve the worst-case behavior of QuickSort.

(b) What advantage does mid-square hashing of numbers have over either the mod or binning methods?

(c) Does the probe function p(k,i) = 3i have any advantages over the probe function p(k,i) = i? Why or why not?

(d) In an application that produced lots of garbage but always had a small active set of data, would mark-sweep or copying collection be the better garbage-collection method? Why?

(e) In what kind of scenario is reference counting better than either garbage-collection method? Why?

(f) From a performance standpoint, what is the main advantage of open hashing over closed hashing

(g) Explain how a hash table implementation can handle key deletion.