Data Structures and Algorithms (CSCI–230)
Prof. Valois — Spring 2000
Exam 1 Answers

1. (a) Write down and solve a summation describing the number of comparisons performed by this function in the average case.

```
template <class T>
bool loop_find(node * list, const T & x) {
  while (list) {
    if (list->data == T) return true;
    list = list->next;
  }
  return false;
}
```

$$\frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n}\frac{n(n+1)}{2} = \frac{n+1}{2}$$

(b) Write down and solve a recurrence describing the number of comparisons performed by this function in the worst case:

```
template <class T>
bool recursive_find(node * list, const T & x) {
  if (!list) return false;
  if (list->data == T) return true;
  return recursive_find(list->next, x);
}
```

$$\begin{aligned} C(n) &= C(n-1)+1 \\ C(n-1) &= C(n-2)+1 \\ &\cdots \\ C(1) &= C(0)+1 \\ C(0) &= 0 \\ \\ C(n) &= n \end{aligned}$$

2. Given the following:

```
template <class T> class binary_tree {
public:
  class iterator {
  public:
    operator bool();
    T & operator*();
```

```
    iterator left();
    iterator right();
    iterator parent();
  };
  iterator root();
};
```

Write generic functions (using templates) to:

(a) Compute the height of a binary tree.

```
template <class Iterator>
int height(Iterator root) {
  if (!root) return -1;
  else return 1 + std::max(height(root.left(), root.right()));
}
```

(b) Find if a given value is in the tree; do not assume the binary search tree invariant. Return true or false.

```
template <class Iterator, class T>
bool find(Iterator root, T x) {
  if (!root) return false;
  else if (*root == x) return true;
  else return find(root.left(), x) || find(root.right(), x);
}
```

(c) Find if a given value is in the tree, this time assuming the binary search tree invariant. Return true or false.

```
template <class Iterator, class T>
bool find(Iterator root, T x) {
  if (!root) return false;
  else if (*root == x) return true;
  else if (*root < x) return find(root.right(), x);
  else return find(root.left(), x);
}
```

3. (a) What is the running time of the vector `push_back` operation? Be as complete as possible.

   Best case: $O(1)$.
   Worst case: $O(n)$, where n is size of vector.
   Amortized case: $O(1)$.

   (b) Provide a piece of code in which `push_back` performs much worse for vector than for list, and explain under what circumstances this happens.

```
a.push_back(b);
```

   If `a` is a list, this will always be $O(1)$. However, if `a` is a vector and `a.size() == a.capacity()`, this will be $O(n)$.

   (c) Provide a piece of code using `push_back` that performs the same with either list or vector.

```
std::vector<int> a;  // or std::list<int>
for (int i=0; i < n; ++i)
  a.push_back(i);
```

Entire loop is always $O(n)$.

4. (a) Why does the list class need to have its own sort function?

   The list class does not provide random access iterators, so the normal generic sorting functions will not work.

   (b) Describe what would happen if the class `std::list` used the default copy constructor assignment operator.

   Whenever a list was copied or assigned from another list, both lists would share the same nodes, since only the internal pointer to the header node would be copied, not the nodes themselves. Thereafter, if nodes were inserted or erased from one list, they would appear and disappear in the other list as well.

   (c) Write the "splice" function below:

```
template <class T> class list {
  struct node {  T data;  node * next;  node * prev; };
  node * head;
public:
  class iterator {
    node * ptr;
  };
  void splice(iterator first, iterator last, iterator pos);
};

template <class T>
void list<T>::splice(iterator first, iterator last, iterator pos)
{
  if (pos != last && first != last)
  {
    last.ptr->prev->next = pos.ptr;
    first.ptr->prev->next = last.ptr;
    pos.ptr->prev->next = first.ptr;
    node * tmp = pos.ptr->prev;
    pos.ptr->prev = last.ptr->prev;
    last.ptr->prev = first.ptr->prev;
    first.ptr->prev = tmp;
  }
}
```