Midterm Review Solutions

Big-O Analysis: Odd-Even Sort

Most of you did well on this question.

(a) OddEvenSort consists of two main loops. The outer loop is the while(TRUE) loop, which continues until no changes are made during on round. Inside the while(TRUE) loop, there are two embedded for loops, each of which loop N/2 times.. Each of those for loops may or may not call the Swap function depending on whether the checked elements are in order or not.

If you look at the Swap function, it has only three assignment statements, which do not depend on the value of n at all. Thus Swap is a O(1) or constant function.

Since the for loops each go N/2 times, they are O(N/2) + O(N/2), which is O(N) + O(N), which is just O(N)..

In the worst case scenario, the while loop will have to iterate N/2 times, making it O(N). Since the O(N) for loops are embedded in the while loop, the worst case running time is O(N) * O(N) = O(N^2).

(b) In the best case scenario, the while loop only goes runs once, and thus is O(1). Then the running time becomes O(1) * O(N) = O(N).

Problem 2 (25 points): Recursion: Your Friends and Boggle!

The simplest way to look at it is: "Write a recursive function to try out all possible assignments of people to the given number of teams. Every time you get a possible arrangement, see if the teams are balanced. If so, you are done, if not, try another arrangement." How to try all possible arrangements? Recursive backtracking. The code is below. The outer function allocates an array to hold the total IQ of each team, initializes the elements to 0, and calls the recursive helper. The helper works like this:

> Is everybody assigned? If so, test to see if the totals are equal.
>
> If everyone is not assigned, try assigning the last person to a team, and make a recursive call to see if that works.
>
> If so, report success.
>
> If not, undo that choice, and try putting the last person somewhere else.
>
> If there are no more choices for that person, report failure.

```
bool CanEvenlySplit(int iQ[], int nPeople, int nTeams)
{
      int *teams, i;

      teams = NewArray(nTeams, int);
      for (i = 0; i < nTeams; i++)
            teams[i] = 0;

      return RecSplit(iQ, nPeople, nTeams, teams);
}

bool RecSplit(int iQ[], int nPeople, int nTeams, int teams[])
{
      int i;

      if (nPeople == 0) return AllEqual(nTeams, teams);

      for (i = 0; i < nTeams; i++)
      {
            teams[i] += iQ[nPeople - 1];
            if (RecSplit(iQ, nPeople - 1, nTeams, teams)) return TRUE;
            teams[i] -= iQ[nPeople - 1];
      }
      return FALSE;
}

bool AllEqual(int n, int array[])
{
      int i;

      for (i = 1; i < n; i++)
      {
            if (array[0] != array[i]) return FALSE;
      }
      return TRUE;
}
```

ADT's Implementation Side - Arrow Keys Galore

(a) Not much to be said about this one. Just look at how the code works...

```
void Insert Character(bufferADT buffer, char ch)
{
      cellT *cp;
      lineT *lp;

      // Create and link in new character as usual
      cp = New(cellT *);
      cp–>ch = ch;
      cp–>link = buffer–>cursor–>link;
      buffer–>cursor–>link = cp;
      buffer–>cursor = cp;

      // If newline, also create a new line and link it
      if (ch == '\n') {
            lp = New(lineT *);
            lp->ln = cp;
            lp->link = buffer->curLine->link;
            buffer->curLine->link = lp;
            buffer->curLine = lp;
      }
}
```

(b) The most common slip up on this one was checking for both **`NULL`** and '**`\n`**' when trying to reposition the cursor on the new line. You could either run into the end of the buffer (**`NULL`**) or hit a **`'\n'`** character. Both would force the cursor to stop there.

```
void MoveCursorDown(bufferADT buffer)
{
      int i, cursorIndex;
      cellT *cp;

      // Do nothing if last line
      if (buffer->curLine->link == NULL) return;

      // Count how far down the cursor is
      for ( cp = buffer->curLine->ln;
            cp != buffer->cursor;
            cp = cp->link) cursorIndex++;

      // Reset pointers appropriately
      buffer->cursor = buffer->curLine->link->ln;
      buffer->curLine = buffer->curLine->link;
      for (i = 0; i < cursorIndex; i++) {
            // Note we must check for both NULL and '\n'.
            // We could run into a '\n' or hit end of the buffer!
            if (buffer->cursor->link != NULL &&
                buffer->cursor->link->ch != '\n')
                  buffer->cursor = buffer->cursor->link;
      }
}
```

ADTs Client Side - Character Buffering

```
void AddData(symtabADT table, string name, string data)
{
      queueADT queue;
      int i;

      queue = Lookup(table, name);
      if (queue == UNDEFINED)
        {
            queue = NewQueue();
            Enter(table, name, queue);
        }

      i = 0;
      while (data[i] != '\0')
            Enqueue(queue, data[i++]);
}

string ExtractData(symtabADT table, string name, int n)
{
      queueADT queue;
      int i;
      string str;

      queue = Lookup(table, name);
      if (queue == UNDEFINED)
        {
            printf("Can't find user %s\n",name);
            return CopyString("");
        }

      if (QueueLength(queue) < n)
            n = QueueLength(queue);
      str = GetBlock((n + 1) * sizeof(char));

      for (i = 0; i < n; i++)
            str[i] = Dequeue(queue);
      str[i] = '\0';
      return str;
}
```