CMPUT 204 (B1) – Assignment # 3

DUE DATE: August 8, 2000

---

1. *Floors and More Floors*

   Let α and β be real numbers, and let n be a positive integer, show that

   (a) $\lfloor \alpha + n \rfloor = \lfloor \alpha \rfloor + n$,

   (b) $\lfloor \alpha + \beta \rfloor - 1 \leq \lfloor \alpha \rfloor + \lfloor \beta \rfloor \leq \lfloor \alpha + \beta \rfloor$,

   (c) $\left\lfloor \frac{\alpha}{n} \right\rfloor = \left\lfloor \frac{\lfloor \alpha \rfloor}{n} \right\rfloor$,

   (d) $\lfloor \alpha \rfloor + \left\lfloor \alpha + \frac{1}{n} \right\rfloor + \left\lfloor \alpha + \frac{2}{n} \right\rfloor + \cdots + \left\lfloor \alpha + \frac{n-1}{n} \right\rfloor = \lfloor n\alpha \rfloor$.

2. *Sorting Out the Bits*

   We assumed so far throughout the course that the keys in a list to be sorted were distinct. However, there are often duplicate keys, and such duplication could make sorting easier, but algorithms that were designed for distinct (or mostly distinct) keys may not take advantage of the duplication. Consider the extreme case, where there are only two possible key values, namely 0 and 1.

   (a) What is the order of the number of key comparisons done by Insertion Sort in the worst case? (Describe a worst-case input).

   (b) What is the order of the number of key comparisons done by Quicksort in the worst case? (Describe a worst-case input).

   (c) Give an efficient algorithm for sorting a list of n keys that may each be either 0 or 1. What is the order of the worst-case running time of your algorithm?

3. *The Selection Problem.*

   In general, the selection problem involves finding the k^{th}- smallest of a set of n distinct integers. An obvious way to find the k^{th}-smallest integer in $\Theta(n \log n)$ time, is to sort the list using quick sort and return the k^{th} entry. In implementation below, we have modified the `split` routine in the quick sort algorithm to find the k^{th}-smallest entry in a list with n entries.

```
#include <stdio.h>

main()
{
    int i,K,N,*B;
    int KthSmallest;
    printf("Enter the number of integers in the list:\n");
    scanf("%d",&N);
    printf("Enter the value of K: (1 <= K <= %d\n",N);
    scanf("%d",&K);
    B = (int *)malloc(N*sizeof(int));
    printf("Enter %d integers:\n",N);
    for (i = 0; i < N; i++)
    scanf("%d",&B[i]);
    KthSmallest = selection(B,0,N,K-1);
    printf("The %d th smallest entry is: %d\n",K,KthSmallest);
    printf("\n");
    return 0;
} /* main */
```

```
int split(int *A, int first, int last)
{   int i, j, pivot, temp;
    pivot = A[first]; j = first;
    for (i = first+1; i <= last; i++)
    {   if (A[i] < pivot)
        {   j++; temp = A[i]; A[i] = A[j]; A[j] = temp;
        }
    }
    temp = A[j]; A[j] = A[first]; A[first] = temp;
    return j;
}/* split */

int selection(int *A, int first, int last, int k)
{
    int position;
    if (first == last) return A[first];
    else
    {   position = split(A,first,last);
        if (k == position) return A[position];
        else if (k < position) return selection(A,first,position-1,k);
        else return selection(A,position+1,last,k);
    }
}/* selection */
```

(a) Explain briefly how the algorithm works.
(b) What is a worst-case input for the algorithm? That is, for what input of list entries and what value of k would the algorithm make the most number of comparisons of list entries?
(c) Let $W(n)$ be the number of comparisons of list entries made in the worst case when *selection* is performed on a list A containing n entries. Find a recurrence relation together with an initial condition satisfied by $W(n)$ for $n \geq 1$. Solve this recurrence relation to find an exact formula for $W(n)$.
(d) Assume that all inputs are equally likely, that is, all values of k are entered with equal frequency and all values of *split* are returned with equal frequency. Let $A(n)$ denote the average number of comparisons of list entries made by *selection* on a list with n entries, find a recurrence relation together with an initial condition satisfied by $A(n)$.
(e) Show that $A(n) \in \Theta(n)$, so that this algorithm does better than quick sort on the average.

4. *Divide and Conquer for Powers.*
A divide and conquer algorithm for computing a^n for two integers a and n is given below.

```
int pow(int a, int n)
{
    if (n == 1) return a;
    if (n%2 == 0)
    { int u;
      u = pow(a,n/2);
      return u*u;
    }
    return a*pow(a,n-1);
}
```

(a) Explain briefly how the algorithm works.
(b) Let $T(n)$ denote the number of multiplications (counting squarings as multiplications) performed by a call to *pow(a,n)*. Find a recurrence relation together with an initial condition satisfied by $T(n)$.
(c) Use the recurrence relation in part (b) to show that $T(n) \in \Theta(\log n)$.