# CS11001/CS11002
# Programming and Data Structures (PDS)
## (Theory: 3-1-0)

More Examples on Recursion

# The Merge Sort Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXSIZE 1000
int A[MAXSIZE];

/* Function prototypes */
void mergeSort ( int, int );
void merge ( int, int, int );
void printArray ( int );
```

# The mergesort function

```
void mergeSort ( int i , int j )
/* i and j are the leftmost and rightmost indices of the current part
   of the array being sorted. */
{
   int mid;

   if (i == j) return;       /* Base case: an array of size 1 is sorted */
   mid = (i + j) / 2;        /* Compute the mid index */
   mergeSort(i,mid);         /* Recursively sort the left half */
   mergeSort(mid+1,j);       /* Recursively sort the right half */
   merge(i,mid,j);           /* Merge the two sorted subarrays */
}
```

# The merge function

```
void merge ( int i1, int j1, int j2 )
{
  int i2, k1, k2, k;
  int tmpArray[MAXSIZE];

  i2 = j1 + 1;
  k1 = i1; k2 = i2; k = 0;
  while ((k1 <= j1) || (k2 <= j2)) {
    if (k1 > j1) {              /* Left half is exhausted */
                                /* Copy from the right half */
      tmpArray[k] = A[k2];
      ++k2;
    } else if (k2 > j2) {       /* Right half is exhausted */
                                /* Copy from the left half */
      tmpArray[k] = A[k1];
      ++k1;
    } else if (A[k1] < A[k2]) {  /* Left pointer points to a smaller value */
                                /* Copy from the left half */
}
```

# The merge function (contd.)

```
tmpArray[k] = A[k1];
      ++k1;
    } else {                    /* Right pointer points to a smaller
   value */
                                /* Copy from the right half */
      tmpArray[k] = A[k2];
      ++k2;
    }
    ++k;                        /* Advance pointer for writing */
  }

  /* Copy temporary array back to the original array */
  --k;
  while (k >= 0) {
    A[i1+k] = tmpArray[k];
    --k;
  }
```

# The print and main functions

```
void printArray ( int s )
{
  int i;

  for (i=0; i<s; ++i) printf("%d ",A[i]);
  printf("\n");
}

int main ()
{
  int s, i;

  srand((unsigned int)time(NULL));
  printf("Array size : "); scanf("%d",&s);
  for (i=0; i<s; ++i) A[i] = 1 + rand() % 99;
  printf("Array before sorting : ");
  printArray(s);
  mergeSort(0,s-1);
  printf("Array after sorting  : ");
  printArray(s);
}
```

# Printing all the subsets of an array

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 26

void printSubset ( int A[], int m );
/* Print the current subset chosen:
   Print 0 as 'a', 1 as 'b', 2 as 'c', and so on. */
{
  int i;

  printf("{");
  if (m > 0) {
    printf("%c", 'a'+A[0]);
    for (i=1; i<m; ++i) printf(",%c",'a'+A[i]);
  }
  printf("}\n");
}
```

# The subset function

```
void subset ( int A[], int n, int m, int k )
/* A stores the choices made so far.
   Subsets of {0,1,2,...,n-1} are printed.
   m is the number of elements chosen in the subset so far.
   k is the last element chosen so far.
   The next choice is allowed to be from k+1,k+2,...,n-1 */
{
   int i; /* i is the next choice */
   printf("n=%d,m=%d,k=%d\n",n,m,k);

   /* Print the subset chosen so far */
   printSubset(A,m);

   /* Choose another element (i), store the choice in the next available
      location of A, and make a recursive call for each choice of i. */
   for (i=k+1; i<n; ++i)
      subset(A,n,m+1,(A[m] = i));
}
```

# The main function

```
int main ()
{
   int A[MAX_SIZE];
   int n;

   printf("Enter n (between 0 and %d): ", MAX_SIZE);
   scanf("%d", &n);
   if ((n < 0) || (n > MAX_SIZE)) exit(1);

   /* First call:
      A is the array for storing the choices.
      n is whatever is supplied by the user.
      No elements are chosen initially (this corresponds to the empty set).
      Last element chosen is -1, so the next element can be from 0 to n-1 */
   subset(A,n,0,-1);

   exit(0);
}
```

Best of luck for MidSems!