Comp. Prog.

Lec 4

Previously

- Recursive programs
    - factorial | fibonacci | n choose k | permutations
- Caching (Memoization/Dynamic Programing)
- Typedef
- Representation/Memory management for Perm
    - Creation/Destruction
- GDB
    - breakpoints | watch expressions | call stack | step over/in/out
- Recursive Drawing
    - recursive circles

Sorting

- Input: an array of integers
- Ouptut: array sorted in increasing order

Pseudo Code: Merge Sort

- Base Case: If array of size 1 then return.
- Otherwise
  - Divide the array into two halfs
  - Copy the halfs into arrays L, R
  - Recursively sort L and R seperately
  - Merge L, R such that the result is sorted

Merge Function

- Input: two arrays L, R that are sorted (seperately)
- Output: an array with size = size(L) + size(R) containing their elements and is sorted
- Example: Merge({1,3,5}, {2,4,6}) should return {1,2,3,4,5,6}

Pseudo Code for Merge

```
t = size(L) + size(R)
A = array of size t
c = 0, l = 0, r = 0;
while(c < t):
    if l reached size(L):
        copy remaining elements from R to A
    if r reached size(R):
        copy remaining elements from L to A
    if L[l] < R[r]:
        A[c++] = L[l++]
    else:
        A[c++] = R[r++]
```

Code for Merge

```
void merge(int *L, int sL, int* R,
           int sR, int *A) {
    int l = 0, r = 0, c = 0;
    while(c <= sL + sR -1) {
        if (r == sR - 1 ) {
            A[c++] = L[l++];
            continue;
        }
        if (l == sL - 1) {
            A[c++] = R[r++];
            continue;
        }
        if (L[l] < R[r]) {
            A[c++] = L[l++];
        } else if (L[l] >= R[r]) {
            A[c++] = R[r++];
        }
    }
}
```

Code for Merge Sort

```
void copy_array(int *A, int start,
                int end, int *B) {
    for(int i = start; i <= end; i++) {
        B[i-start] = A[i];
    }
}
void sort(int *A, int len) {
    if (len == 1) {
        return;
    } else {
        int mid = len/2;
        int L [ mid], R [len - mid];
        copy_array(A, 0, mid, L);
        copy_array(A, mid, len, R);
        sort(L, mid);
        sort(R,len - mid);
        merge(L, mid, R, len-mid, A);
    }
}
```

Doesnt seem to work

Debug and fix!

Yet another recursive algo

For sorting an array A from `start` to `end`

- Base case: `start == end` then nothing to be done.
- Find the index `i` of the smallest element in A from `start` to `end`
- swap the values of `A[start]` and `A[i]`
- Sort `A` from `start + 1` to `end`

HW: Implement it.

Why should you pass pointers instead of arrays?

Why should you pass pointers instead of arrays?

- C passes arguments **by value**.
- Every time an array is passed, entire element is copied.
- Also modifications done to the Array will not be saved.

Passing pointers

- Passing pointers, only results in copying of the pointer(ie address of the first element of the array).
- Also modifications will remain, as we are changing the actual memory location.