CSC180 tutorial notes – week of Oct. 6

We cover some pass by value/pass by reference stuff, and then continue with some algorithms dealing with arrays.

Just cover the items in order.

Pass by value/pass by reference

Remember the problem from the tutorial two weeks ago where we wanted to solve the quadratic equation?

```c
int quadratic_roots(double a, double b, double c, double roots[])
{
    double disc = b * b - 4 * a * c;
    if(disc < 0)
        return -1;
    roots[0] = (-b + sqrt(disc))/(2 * a);
    roots[1] = (-b - sqrt(disc))/(2 * a);
    return 0;
}
```

Note: quadratic_roots(double a, double b, double c, double root1, double root2) wouldn’t do the trick, since changing root1 in such a function wouldn’t affect the value in the caller function.

Remind them to know this distinction.
**Algorithms**

Warm-up: write a function to find the max of an array of ints, and the index of the max element

```c
void arr_max(int arr[], int n, int ans[])
{
    int i;
    int cur_max = arr[0];
    int cur_max_ind = 0;
    for(i = 1; i < n; i++){
        if(cur_max < arr[i]){
            cur_max = arr[i];
            cur_max_ind = i;
        }
    }
    ans[0] = cur_max;
    ans[1] = cur_max_ind;
}
```

Return the three highest elements of an array of ints
Simple idea: remove the highest element (possibly by setting it to –INT_MAX-1), then find the new highest element

Better idea: keep the current top three elements, sorted.

```c
void consider_new(int top3[], int new_elem)
{
    if(new_elem < top3[0])
        return;
    if(new_elem < top3[1])
        top3[0] = new_elem;
    else if(new_elem < top3[2]){
        top3[0] = top3[1];
        top3[1] = new_elem;
    }else{
        top3[0] = top3[1];
        top3[1] = top3[2];
        top3[2] = new_elem;
    }
}
```
void arr_max3(int arr[], int top3[], int n)
{
  int i;
  for(i = 0; i < 3; i++){
    top3[i] = -INT_MAX-1;/*smallest int*/
  }
  for(i = 0; i < n; i++){
    consider_new(top3, arr[i]);
  }
}

Sort an array of ints in non-decreasing order

void max_sort(int arr[], int n)
{
  int i;
  int max_res[2];
  int cur_max;
  for (i = 0; i < n; i++){
    arr_max(arr, n-i, max_res);
    arr[max_res[1]] = arr[n-i-1];
    arr[n-i-1] = max_res[0];
  }
}

Given a string(i.e., array of chars) and its length, determine whether it’s a palindrome, 
ignoring spaces and uppercase/lowercase

A palindrome is a string which is the same read forward and backward

Example of a palindrome: "A man, a plan, a canal—Panama!"

(aside: remember that chars are basically numbers. All we need to know is that a…z and 
A…Z correspond to sequential numbers. Specifically, in ASCII, A-Z is 33-58 and a-z is 
65-90)
char to_lower_case(char c)
{
    if( (c >= 'A') && ( c <= 'Z' ) )
        return c+'a'-'A';
    else
        return c;
}

int is_letter(char c)
{
    return ( (c >= 'a')&&(c <= 'z') ) || ( (c >= 'A')&&(c<='Z') );
}

int is_palindrome(char str[], int n)
{
    int i = 0, j = n - 1;

    while(i < j){
        if(!is_letter(str[i]))
            i++;
        else if(!is_letter(str[j]))
            j--;
        else if(to_lower_case(str[i]) != to_lower_case(str[j]))
            return 0;
        else{
            i++; j--;
        }
    }
    return 1;
}
Reverse a string in place

void reverse(char str[])
{
    int i;
    int l = strlen(str);
    for(i = 0; i < l/2; i++){
        char temp;
        temp = str[i];
        str[i] = str[l-i-1];
        str[l-i-1] = temp;
    }
}