Rick Sullivan
Professor Tran
Algorithms-COEN 179
30 April 2013

Homework # 2

1. (a) **Quick sort:**

1_a	1_b	1_c

Begin with different elements of same sort value.

1_a	1_b	1_c

Do nothing as you move through the array (each element = pivot).

1_c	1_b	1_a

At the end, swap i and the pivot.

Now ordering is not preserved.

(b) **Heap sort:**

1_a	1_b	1_c

Begin with different elements of same sort value.

1_a	1_b	1_c

Construct the heap (which does nothing in this case).

1_c	1_b	1_a

Swap the largest element with the last, and now repeat while ignoring it.

1_c	1_b	1_a

Reconstruct heap (do nothing).

1_b	1_c	1_a

Swap again.

1_b	1_c	1_a

Done. Ordering is not preserved.

2.
```
#include <iostream>
#include <algorithm>
#include <ctime>
#include <cmath>

using namespace std;

template <class Item>
void fix_heap(Item H[], size_t i, size_t n)
{
    while ((2*i) + 1 <= n )
    {
        size_t largest = i;
```

```
        size_t left = 2*i+1;
        size_t right = left + 1;

        largest = (H[i] >= H[left]) ? i : left;
        if (right <= n && H[right] > H[largest])
            largest = right;
        if (largest == i)
            return;
        swap(H[i], H[largest]);
        i = largest;
    }
}

template <class Item>
void make_heap(Item H[], size_t n)
{
    for (size_t i = ((n-1)/2); i >= 0; --i)
        fix_heap(H, i, n);
}

template <class Item>
void heap_sort(Item H[], size_t n)
{
    make_heap(H, n-1);
    for (size_t i = n-1; i > 0; --i)
    {
        swap(H[0], H[i]);
        fix_heap(H, 0, i);
    }
}


int main()
{
    srand(time(0));

    size_t n;
    cout << "Enter n: ";
    cin >> n;

    int H[n];

    for (size_t i = 0; i < n; ++i)
        H[i] = i;

    for (size_t i = n-1; i > 0; --i)
    {
        size_t j = (rand() % n);
        swap(H[i], H[j]);
    }

    for (size_t i = 0; i < n; ++i)
        cout << H[i] << " ";
    cout << endl;
```

```
    heap_sort(H, n);

    for (size_t i = 0; i < n; ++i)
    {
        cout << H[i] << " ";
        if (H[i] != i)
        {
            cout << "Error!";
            exit(1);
        }
    }
    cout << "Sorted!" << endl;
}
```

3.

```
//Program to find the most frequently occurring element
//in an array.
#include <iostream>
#include <algorithm>
#include <vector>
#include <ctime>

void vPrint(std::vector<int> &V){
    for(int i = 0; i < V.size(); i++)
        std::cout << V[i] << ' ';
    std::cout << std::endl;
}

void fillVector(std::vector<int> &V, int range, int num){
    for(int i = 0; i < num; i++){
        int n = rand()%range;
        V.push_back(n);
    }
}

//Outputs the element that occurs the most and the number of times.
//If more than one element occur the same number of times, the lower
//element is chosen.
int main(){
    srand(time(0));
    std::vector<int> A;
    fillVector(A, 20, 500);
    vPrint(A);
    std::stable_sort(A.begin(), A.end());    //nlgn time
    unsigned maxCount = 0;
    unsigned currentCount = 0;
    unsigned elementIndex = 0;
    unsigned maxIndex = 0;
    for(int i = 0; i < A.size(); i++){  //Linear time
        if(A[i] != A[elementIndex]){ //If we are looking at a new int
            currentCount = 0;
            elementIndex = i;
        }
        currentCount++;
```

```
            if(currentCount > maxCount){
                maxCount = currentCount;
                maxIndex = elementIndex;
            }
        }
        std::cout << A[maxIndex] << " occurs " << maxCount
        << " times." << std::endl;
        return 0;
}
```