EECE 7205-Assignment 2

Sreejith Sreekumar: 001277209

October 11, 2018

1 Randomized Quicksort

1.1 Code

```
#include <iostream>
#include <cstdlib>
#include <bits/stdc++.h>
#include <time.h>

using namespace std;

int randomPartition(int* data, int begin, int end){

    srand(time(NULL));

    /*
     * Random pivot index
     */
    int pivot_index = begin + rand() % (end-begin+1);
    int pivot = data[pivot_index];
    swap(data[pivot_index], data[end]);

    pivot_index = end;
    int i = begin - 1;

    for(int j=begin; j<=end-1; j++)
    {
        if(data[j] <= pivot)
        {
            i = i+1;
            swap(data[i], data[j]);
        }
    }

    swap(data[i+1], data[pivot_index]);
    return i+1;
}
```

```
void randomQuickSort(int* data, int start, int end)
{
    if(start < end) {
        int mid = randomPartition(data, start, end);
        randomQuickSort(data, start, mid-1);
        randomQuickSort(data, mid+1, end);
    }
}


int main(){

  int input[100];

  for(int i=0;i<100;i++){
    input[i] = i+1;
  }

  clock_t start = clock();
  randomQuickSort(input, 0, 100);
  clock_t end = clock();

  cout<<"After sorting: ";
   for(int i=0;i<100;i++){
     cout<<input[i]<<"\t";
   }
  cout<<"\n";

  double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

  printf("Randomized Quicksort took %f seconds to finish \n", cpu_time_used);
}
```

1.2 Running Times

Five run times recorded for sorting using the randomized quicksort algorithm above were as follows:

- 0.000246 seconds
- 0.000269 seconds
- 0.000082 seconds
- 0.000284 seconds
- 0.000198 seconds

2 Heapsort

```
#include <iostream>
#include <cstdlib>
#include <algorithm>


using namespace std;

void print_array(int arr[], int size) {
  for(int i = 0 ; i < size; i++) {
    cout << arr[i] << "\t";
  }
  cout << endl;
}

void max_heapify(int array[], int i, int n) {
  if (i < 0) {
    return;
  }
  int l = 2 * i + 1;
  int r = 2 * i + 2;
  int largest_index = i;
  if( l < n && array[l] > array[i]) {
    largest_index=l;
  }
  if( r < n && array[r] > array[largest_index]) {
    largest_index=r;
  }
  if(largest_index != i) {
    int temp = array[i];
    array[i] = array[largest_index];
    array[largest_index] = temp;
  }
  max_heapify(array, i - 1, n);
}


void heapsort(int array[], int size){
  cout << endl;
  for (int i = size-1; i > 0; i--) {
      max_heapify(array, i/2, i+1);
      int temp = array[i];
      array[i] = array[0];
      array[0] = temp;
    }
}
```

```
int main(){

  int size = 100;

  int input[size];

  for(int i=0;i<100;i++){
    input[i] = i+1;
  }

  /*
   * Create a random permutation of the numbers in the input array
   */
  random_shuffle(&input[0], &input[size-1]);

  cout<<"\nNumbers to be sorted (1 to 100) in a random permutation: \n";
  print_array(input, size);

  heapsort(input, size);

  cout<<"\n Heapsort Output: \n";
  print_array(input, size);
  std::cout<<"\n";
}
```

3 Counting Sort

```
#include <iostream>


int findMax(int* data, int limit){

  int max = data[0];
  for(int i=0; i<limit; i++){
    if(data[i] > max){
      max = data[i];
    }
  }
  return max;

}


void sort(int* data, int limit){

  int largest = findMax(data, limit);
```

```
  /*
    Loop 1: Create an array with limit - the largest element
  */
  int _counts[largest+1] = {0};

  /*
    Loop 2
   */
  for(int i=0; i<limit; i++){
    _counts[data[i]]+=1;
  }

  /*
    Loop 3
   */
  for(int i=2; i<=largest; i++){
    _counts[i] += _counts[i-1];
  }


  int B[limit] = {0};
  for(int j=limit-1; j>=0; j--){
    B[_counts[data[j]]] = data[j];
    _counts[data[j]] = data[j] - 1;
  }


  std::cout<<"\nSorted Array:";
  for(int i=0; i<limit; i++){
    std::cout<<B[i] << "\t";
  }

  std::cout<<"\n";
}



int main( int argc, char* argv[] )
{

    int A [] = {20, 18, 5, 7, 16, 10, 9, 3, 12, 14, 0};
    int limit = 11;

    std::cout<<"Input Array: ";
    for( int i = 0; i < limit; i++ ){
      std::cout<<A[i] << "\t";
    }
```

```
    sort(A, limit);

    return 0;

}
```

4 Radix Sort

```
#include<iostream>


using namespace std;

int get_greatest_element(int data[], int limit) {

        int greatest = data[0];
        for (int i = 1; i < limit; i++)
                if (data[i] > greatest)
                        greatest = data[i];
        return greatest;
}

/*
 * Counting sort for the sorting the decimal digits
 */
void countingSort(int data[], int limit, int exp){

        int output[limit];
        int i, count[10] = {0};

        for (i = 0; i < limit; i++)
                count[(data[i]/exp)%10 ]++;

        for (i = 1; i < 10; i++)
                count[i] += count[i - 1];

        for (i = limit - 1; i >= 0; i--)
        {
                output[count[ (data[i]/exp)%10 ] - 1] = data[i];
                count[ (data[i]/exp)%10 ]--;
        }

        for (i = 0; i < limit; i++)
                data[i] = output[i];
```

```
}

void radixsort(int data[], int limit){

  int greatest_element = get_greatest_element(data, limit);

  for (int base = 1; greatest_element/base > 0; base *= 10) {
    countingSort(data, limit, base);
  }

}

int main() {

     int input[] = {329, 457, 657, 839, 436, 720, 353};
     int limit = sizeof(input)/sizeof(input[0]);

     radixsort(input, limit);

    for(int i=0;i<limit;i++)
      std::cout<<input[i]<<"\t";

     return 0;
}
```