Adam Frazee
CSE 464
Gregory Gelfond
10/4/2015

Mergesort Algorithm

History: The mergesort algorithm was invented by John von Neumann in 1945. This divide and conquer algorithm takes an unsorted data structure, and then returns one that is sorted. The big O cost of sorting in this fashion is $O(nlog(n))$. To break down this complexity we must understand how the algorithm sorts. Mergesort first splits your data structure into pieces until you have sub pieces of 1. By definition we will consider these pieces sorted. The operation of splitting costs $O(log(n))$. Then it must compare items while combining. The comparison operation cost $O(n)$ since you are comparing every in the data structure. It is important to note that mergesort will keep this complexity for any data size. This is in comparison to quick sort. Which has an average case of $O(nlog(n))$. However depending on the way data is sorted, and the implementation of that algorithm we get a best case of $O(n)$ and a worst case of $O(n^2)$.

Formulating the problem: Given a group A of 0 to N objects. We would like to sort them in a way that $\forall\ i, j\ 0 \leq i < j < N\ S.TA[i] \leq A[j]$.

Pseudo Code:

Algorithm 1 Mergesort

```
1:  procedure MERGESORT(A[])
2:      if A[].size() == 1 then return A[]
3:      while (A[].size()  > 1) do
4:          left is an array with range [0,N/2] //floor math
5:          right is an array with range [(N/2) + 1, N]
6:
7:          left = mergesort(left)
8:          right = mergesort(right)
9:
10:         return merge(left,right)
11: procedure MERGE(A[],B[])
12:     C[] is an empty array
13:     while (A[] && B[] has elements) do
14:         if (A[0] < B[0]) then
15:             place B[0] at the end of C
16:             remove B[0]
17:         else
18:             add A[0] to the end of C[]
19:             remove A[0] from A[]
20:         while (A[].size() != 0) do
21:             add A[0] to the end of C[]
22:             remove A[0] from A[]
23:         while (B[].size() != 0) do
24:             add B[0] to the end of C[]
25:             remove B[0] from B[]
```

Loop Invariant: While all arrays split down to size one . We look at k which is an element in A[]. When merging and placing elements we know that the smallest element will always be on the left. Since this happens at the base case. You can assume that all elements will sort themselves from right(least) to left(most).

Formulating Correctness:

Base Case: $N = 1$ an array of this length is sorted by definition.
Inductive Hypothesis:

Assume that the elements n=0,1,2,3...k are sorted in ascending order.

Inductive Step:

So does mergesort sort when $k \rightarrow k + 1$ still return a sorted array. So we have an array of A[0] to A[k+1]. This array is divided into two halves. All the way down to our base case.We have proved that our base case is sorted. Thus making it valid.

Code: main.cpp

```
#include <stdlib.h>
#include <list>
#include <string>
#include <iostream>
#include "main.h"
#include <time.h>       /* time */

int main(){
  std::list<int> return_list, list;
  std::string logic;
  std::list<int>::iterator it;
  std::cin >> logic;

  if (logic == "create"){
    return_list = input_list();
  }
  if (logic == "random"){
    return_list = random_list();
  }
  std::cout << "The inut contains:";
  for (it = return_list.begin(); it != return_list.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  list = mergesort(return_list);

  std::cout << "The return contains:";
  for (it = list.begin(); it != list.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';
  bool flag = true;
  system("PAUSE");
  return 0;
}
std::list<int> input_list(){
  std::list<int> r_list;
  int a;
  while (std::cin >> a){
    r_list.push_back(a);
  }
  return r_list;


}

std::list<int> random_list(){
  std::list<int> list;
  srand(time(NULL));
  int rand_num = rand()%50;

  for (int i = 0; i < rand_num; i++){
    int push_num = rand() % 9999999;
    list.push_back(push_num);
  }
  return list;
}

std::list<int> mergesort(std::list<int> & list_a){
  std::list<int> left,right;
  std::list<int>::iterator it;
  int size = list_a.size();
  //spilt down to a certain size.
```

```
63    it = list_a.begin();
64    if (list_a.size() == 1){
65      return list_a;
66    }
67    while (list_a.size() > 1){
68      for (int i = 0; i < size / 2; i++){
69        left.push_back(list_a.front());
70        list_a.pop_front();
71      }
72      left = mergesort(left);
73      right = mergesort(list_a);
74
75      return merge(left,right);
76    }
77
78
79  }
80
81  std::list<int> merge(std::list<int>& a ,std::list<int>& b){
82    std::list<int> list_c;
83    std::list<int>::iterator it_a, it_b;
84    it_a = a.begin();
85    it_b = b.begin();
86    while (a.size() != 0 && b.size() != 0){
87      if (a.front() < b.front()){
88        list_c.push_back(a.front());
89        a.pop_front();
90      }
91      else{
92        list_c.push_back(b.front());
93        b.pop_front();
94      }
95    }
96    while (a.size() !=0){
97      list_c.push_back(a.front());
98      a.pop_front();
99    }
100   while (b.size() != 0){
101     list_c.push_back(b.front());
102     b.pop_front();
103   }
104   return list_c;
105 }
```

main.h

```
1  #include <stdlib.h>
2  #include<list>
3  #ifndef MAIN_H_
4  #define MAIN_H_
5
6  std::list<int> mergesort(std::list<int>& list_a);
7  std::list<int> merge(std::list<int>& a, std::list<int>& b);
8  std::list<int> random_list();
9  std::list<int> input_list();
10
11 #endif
```

How to Compile: I will be uploading the complete Microsoft visual studio solution file. You can open on any Benton lab and compile there. If you are running on windows you can take the compiled .exe and run it. It will not run on linux or Mac because of a system call used at the end of main.