**Bowen Wang**

**301267523**

**CMPT 125**

**Assignment 3**

**Question 1**

a)

void merge(int arr[], int start, int mid, int end);

void mergesort(int arr[],int len){

int i,j;

for (i=1; i<=len-1; i\*=2){

for(j=0; j<len-1; j+=2i){

int mid=j+i-1;

int right\_end;

if(j+2\*i-1<len-1){

right\_end=j+2\*i-1;

}else{

right\_end=len-1;

}

merge(arr, j, mid, right\_end);

}

b)

int \*merge\_two(int arr1[], int arr2[], int len1, int len2){

int i=0,j=0,k=0;

int new\_arr[len1+len2];

while (i<len1 && j<len2){

if(arr1[i]<arr2[j]){

new\_arr[k]=arr1[i];

k++;

i++;

}else{

new\_arr[k]=arr2[j];

k++;

j++;

}

}

while(i<len1){

new\_arr[k]=arr1[i];

k++;

i++;

}

while (j<len2){

new\_arr[k]=arr2[j];

j++;

k++;

}

return new\_arr;

}

Question 2

a)

void overflow(int i){

if(i==INT\_MAX){

printf(“%d\n”, i);

} else {

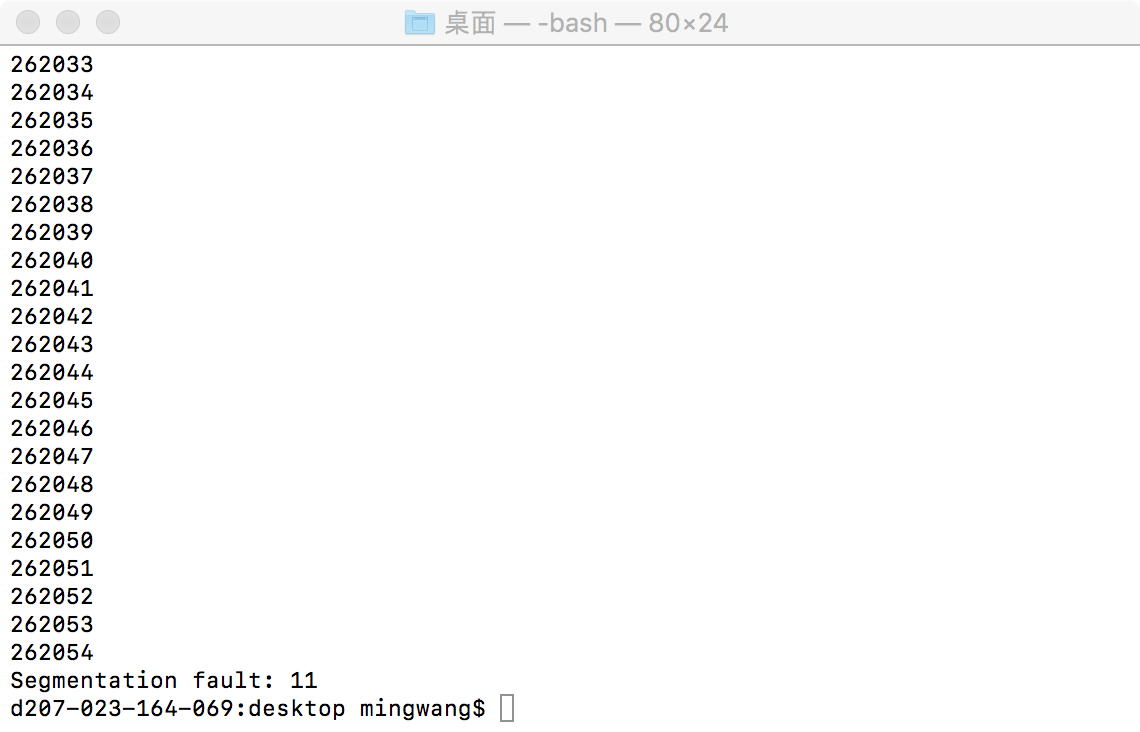
printf(“%d\n”, i);

overflow(i+1);

}

}

b)



c)

d)

For the first function foo(), the array is defined by malloc, which is point to a address in head memory. But the second function bar(), the array is not using dynamic memory, so it is pointed to an address in stack memory. The head memory is much larger than stack memory in a computer, so the first function can go much further than the second one.

e)

As the stack memory is 1MB=1024KB=1024\*1024byte, the max time of recursive calls is 1024\*1024/256=4096 (times), each time of recursive call, the array is divided into 2 parts, assume an array can be divided 4096 times, the minimum size of the array should be 2^4096, when the array size is 2^4096, it already cause to a heap overflow, so a heap overflow always happen before a stack overflow.

Question 3

1. Pick the first case out, set it as a pivot and compare every rest case number with the picked one, if the case number is smaller than the pivot, put the case on the left of the pivot, if the case number is larger than the pivot case number, put it to the right of the pivot. After that we got two heaps of cases, the heap on the left contains all cases smaller than the pivot and the heap on the right contains all cases larger than the pivot. Then pick the first case from a heap and set it as the second pivot, compare every case in that heap with the picked pivot, and separate the heap to two parts, one is smaller than pivot and the other is larger than the pivot, repeat the process until every case is separated, after that, the cases can be sorted from right to left.

void quick\_sort(int arr[],int start,int end){

if start<end;

int pivot=start; //identify the position of the pivot point.

set int i=start +1;

while (i<=end){

if (arr[i]<arr[pivot]){

int temp =arr[pivot]；

arr[pivot]=arr[i];

arr[i]=temp; // put the document on the left of the pivot

pivot = pivot +1;

}else {

//put the document on the right of the pivot;

}

i++;

}

quick\_sort ( arr , start , pivot -1 ); // repeat the process to the left part.

quick\_sort ( arr, pivot+1, end); // repeat the process to the right part

}

b)

The big O of quick sort under average situation is O(nlogn), so in this case, n=1000, the running time is 1000\*log(1000).

The best case and also average case for Bob is every pivot he pick is in the middle of the part. In this case, the time of comparison Bob should do is 1000\*log(1000)=9970 (times). Assuming Bob spends 5 second to compare, and 0.5s to pick up a file, the total time is 9970\*5.5=54835 s.

The worst case for Bob is every pivot is the smallest or largest in the part, so that the prime part is divide to one part instead of two. In this case, the time of comparison Bob should do is 1000\*1000=1,000,000 (times). Assuming Bob spends 5 second to compare, and 0.5s to pick up a file, the total time is 1,000,000\*5.5=5,500,000 s.