Bowen Wang

301267523

CMPT 125

SUMMER 2017

**Assignment 2**

Question 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Selection Sort | Sorted | Reversed | Random |
| 20,000 | 0.401s | 0.446s | 0.429s |
| 40,000 | 1.496s | 1.734s | 1.675s |
| 80,000 | 5.916s | 7.05s | 6.778s |
| Insertion Sort | Sorted | Reversed | Random |
| 20,000 | 0.005s | 0.552s | 0.292s |
| 40,000 | 0.005s | 2.192s | 1.109s |
| 80,000 | 0.005s | 8.917s | 4.332s |

The running time of selection sorting is generally longer than Insertion Sort except for reversed data. And for data type, sorted data is the easiest to sort, while reversed is hardest. Selection sort for sorted data is O(n2), Selection sort for reversed data is O(n2), Selection sort for random data is O(n2). Insertion sort for sorted data is O(n), Insertion sort for reversed data is O(n2), Insertion sort for random data is O(n2)

Question 2.

1. void swap (int arr[], int i, int j){

int temp;

temp = arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

1. void ssort\_down(int arr[], int n){

for (int i=n; i>1; i--){

int largest=I;

for (int j=I; j>0; j--){

if (arr[j]>arr[largest]){

largest=j;

}

}

int temp=arr[i];

arr[i]=arr[largest];

arr[largest]=temp;

}

}

1. int max(int arr[], int start, int end){

if(start==end){

return start;

} else {

int maximum= max( arr, start+1, end);

if(arr[start]>arr[maximum]){

maximum=start;

}

return maximum;

}

}

1. int rec\_ssort(int arr[],int n){

if(n>1){

swap(arr, max(arr,0,n), n);

rec\_ssort(arr,n-1);

}

}

Question 3

1. // post-condition: return the smallest i >= 0 such that arr[i]==target; or if target is not found in arr, -1 is returned.
2. assert(i>=0 && i<n);

// assertion: i>=0; i<n; the target is not found in first i element(s), and the for loop is going to check whether the (i+1)th element is the target or not, if yes, i is returned, if not, i is going to increase by 1 and re-run the for loop.

1. Initialization: when i==0, the first element in the array is taken to check with target, which satisfies the process of checking the existence of the target.

Maintenance: at the start of loop i, target is not found in i elements before. After the loop iteration, if arr[i]==target, i would be returned, if arr[i]!=target, the loop continues till i=n.

Termination: when i=n, every element in arr has been checked and not pair with target, so the for loop is jumped out and return -1, which indicates the target is not found in arr.

Question 4

1. The algorithm is Binary Search.
2. O(n)
3. The O notation running time of insertion and removal algorithm is O(n). To identify the position to insert or remove, running time is O(log2n), then the pulling work running time is O(n), so O(log2n)+O(n)=O(n).
4. The running time of selection sorting is O(n2), the running time of insertion sorting is also O(n2), but when we consider that the data is sorted at the beginning of every week, the running time of selection sorting for sorted data is O(n2), while the running time of insertion sorting for sorted data is O(n), when n>1, n2>n, so I assume the files is mostly sorted at the end of week and suggest insertion sorting.

Question 5

//Pre-condition: all n elements in arr should be int, the value range is between 0 and n-1.

void ssort(int arr[], int n){

initialize int i, j, k, m as zero;

initialize int count\_arr[n] as zero; // set a new array to save the time of number occurred, each element in count\_arr represent the respective number, e.g. count\_arr[0] represent the time of “0”appears in arr.

While the index i is less than the size of arr. // count how many time does the number i occur in arr

set value of count\_arr[arr[i]] += 1;

While the index j is less than the size of arr.

While the index k is less than the size of (count\_arr[j]) //reset the values in arr

Set the value of the element in arr at position m as j;

Set the value of m += 1;

}

