電機系三年級

盧柏儒

March 26, 2015

Algorithm PA1 Report

In this PA1, my algorithm is followed by the textbook 'introduction of Algorithm'.

Insertion sort

```
// Insertsion sort method
void SortTool::InsertionSort(vector<int>& data) {
    // Function : Insertion sort
    // TODO : Please complete insertion sort code here

    for(int j=1;j<data.size();j++){
        int key=data[j];
        int i=j-1;

        while(i>=0 && data[i]>key){
            data[i+1]=data[i];
            i=i-1;
        }
        data[i+1]=key;
    }

}
```

- Merge sort

```
// Merge sort method
void SortTool::MergeSort(vector<int>& data){
    SortSubVector(data, 0, data.size() - 1);
}

// Sort subvector
void SortTool::SortSubVector(vector<int>& data, int low, int high) {
    // Function : Sort subvector
    // TODO : Please complete SortSubVector code here
    // Hint : recursively call itself
    //        Merge function is needed
    if(low < high){
        int middle1 = floor((high + low)/2);
        int middle2 = middle1+1;
        SortSubVector(data, low, middle1);
        SortSubVector(data, middle2, high);
        Merge(data, low, middle1, middle2, high);
    }
}
```

```
// Merge
void SortTool::Merge(vector<int>& data, int low, int middle1, int middle2, int high)
 {
    // Function : Merge two sorted subvector
    // TODO : Please complete the function
    int n1 = middle1-low+1;
    int n2 = high-middle2+1;
    int* L = new int [n1+1];
    int* R = new int [n2+1];

    for(int i=0; i<n1; ++i)
        L[i] = data[low+i];
    for(int j=0; j<n2; ++j)
        R[j] = data[middle2+j];
    L[n1] = R[n2] = INT_MAX;

    int ptr_L = 0, ptr_R = 0;

    for(int k=low; k<=high; k++){
        if(L[ptr_L] <= R[ptr_R]){
            data[k] = L[ptr_L];
            ++ptr_L;
        }
        else{
            data[k] = R[ptr_R];
            ++ptr_R;
        }
    }
    delete [] L;
    delete [] R;
}
```

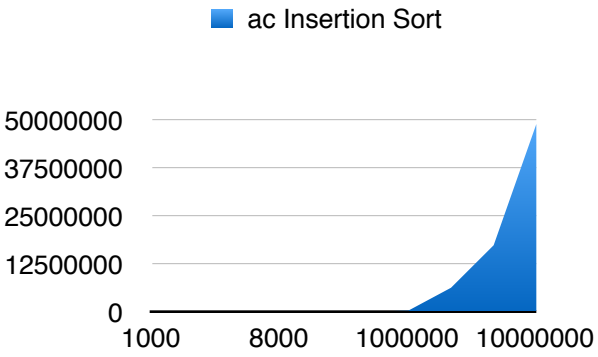
In the merge sort, I include the <limit.h> library and use the INT_MAX as my sentinel.

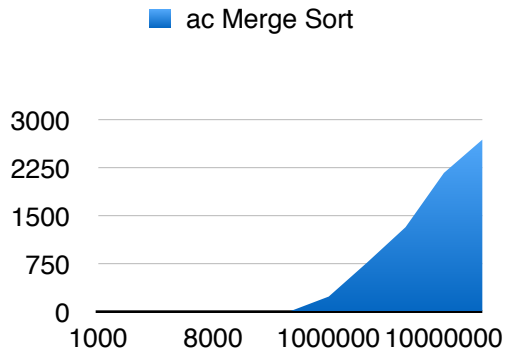
Heap sort

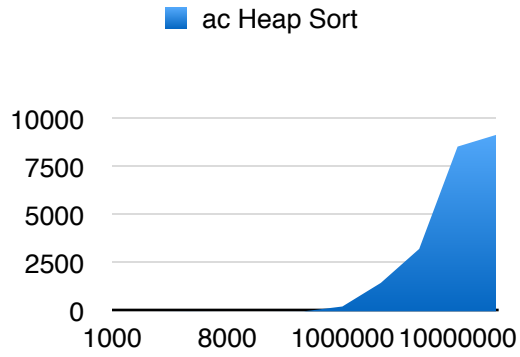
```
//Max heapify
void SortTool::Max_Heapify(vector<int>& data, int root) {
    // Function : Make tree with given root be a max-heap if both right and left sub
-tree are max-heap
    // TODO : Please complete max-heapify code here
    int left = root*2+1;
    int right = root*2+2;
    int largest;

    if(left<heapSize && data[left]>data[root])
        largest = left;
    else
        largest = root;
    if(right<heapSize && data[right]>data[largest])
        largest = right;
    if(largest != root){
        swap(data[root], data[largest]);
        Max_Heapify(data, largest);
    }
}

//Build max heap
void SortTool::Build_Max_Heap(vector<int>& data){
    heapSize=data.size();
    for(int i=floor(heapSize/2)-1; i>=0; --i)
            Max_Heapify(data,i);
}
```

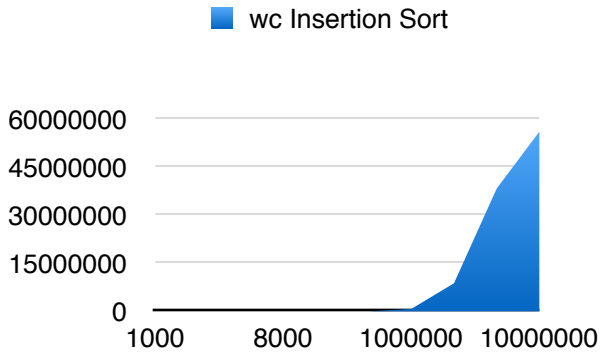
The table

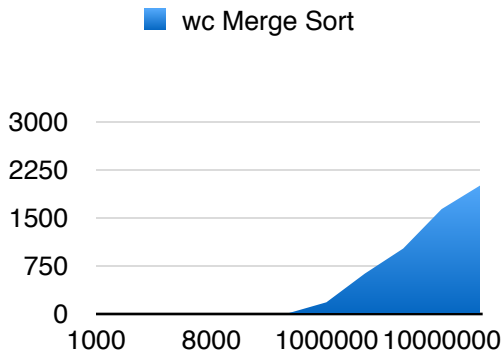
Input size	IS		MS		HS	
	CPU time (ms)	Memory (KB)	CPU time (ms)	Memory (KB)	CPU time (ms)	Memory (KB)
1000.bc	0	12592	0	12592	0	12592
1000.wc	0	12592	0	12592	0	12592
1000.ac	0	12592	0	12592	0	12592
4000.bc	0	12592	0	12592	0	12592
4000.wc	8	12592	0	12592	4	12592
4000.ac	4	12592	0	12592	4	12592
16000.bc	4	12736	4	12736	0	12376
16000.wc	100	12736	4	12736	0	12376
16000.ac	52	12736	4	12736	0	12376
32000.bc	0	12736	4.001	12736	4	12736
32000.wc	412.026	12736	4	12736	4	12736
32000.ac	196.012	12736	8.001	12736	4	12736
1000000.bc	4	18756	168.01	20612	172.011	18756
1000000.wc	754415	18756	176.011	20612	156.01	18756
1000000.ac	273449	18756	236.014	20612	252.016	18756
10000000.bc	28.002	110916	1580.1	117208	2168.14	110916
10000000.wc	56160000	110916	2000.12	117208	1972.12	110916
10000000.ac	48859200	110916	2688.17	117208	9188.57	110916
2000.bc	0	12592	0	12592	0	12592
2000.wc	4	12592	0	12592	0	12592
2000.ac	0	12592	0	12592	0	12592
3000000.bc	12.001	37188	556.034	40712	588.036	37188
3000000.wc	8782360	37188	624.038	40712	532.034	37188
3000000.ac	6228010	37188	768.048	40712	1480.09	37188
5000000.bc	12.001	61764	1064.07	64912	932.059	61764
5000000.wc	38351500	61764	1016.06	64912	880.055	61764
5000000.ac	17302400	61764	1320.08	64912	3252.2	61764
8000000.bc	20	61764	1580.1	76628	1640.1	61764
8000000.wc	>1hr	61764	1632.1	76628	1592.1	61764
8000000.ac	>1hr	61764	2168.13	76628	8580.54	61764
8000.bc	0	12592	4	12592	0	12592
8000.wc	24	12592	0	12592	0	12592
8000.ac	12	12592	4	12592	0	12592

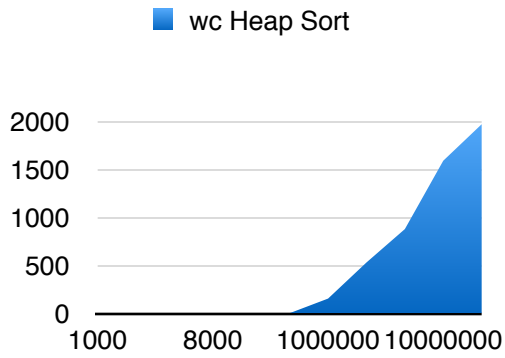

ac Insertion Sort
50000000
37500000
25000000
12500000
0
1000
8000
1000000
10000000

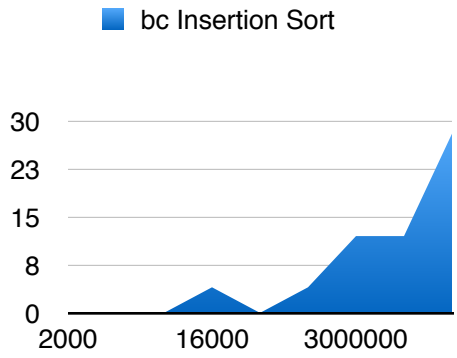

ac Merge Sort
3000
2250
1500
750
0
1000
8000
1000000
10000000

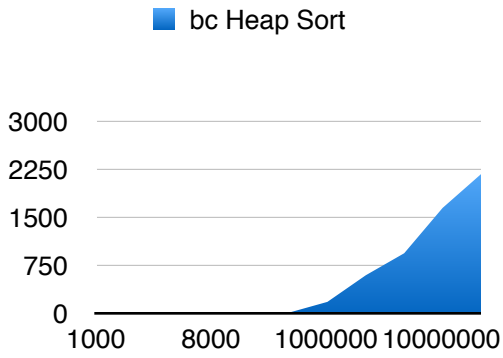

ac Heap Sort
10000
7500
5000
2500
0
1000
8000
1000000
10000000

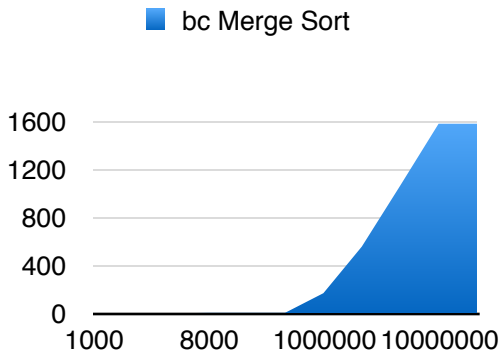

wc Insertion Sort
60000000
45000000
30000000
15000000
0
1000
8000
1000000
10000000


wc Merge Sort
3000
2250
1500
750
0
1000
8000
1000000
10000000


wc Heap Sort
2000
1500
1000
500
0
1000
8000
1000000
10000000


bc Insertion Sort
30
23
15
8
0
2000
16000
3000000


bc Heap Sort
3000
2250
1500
750
0
1000
8000
1000000
10000000


bc Merge Sort
1600
1200
800
400
0
1000
8000
1000000
10000000