Sorting Algorithms Analysis

Ulises Méndez Martínez

Design and Analysis of Algorithms 2016-01

ulisesmdzmtz@gmail.com

March 14, 2016

Overview

1. Merge Sort
   - Implementation
2. Heap Sort
   - Implementation
3. Quick Sort
   - Implementation
4. Comparison
5. Comparison

Merge Sort

Is an efficient, general-purpose, comparison-based sorting algorithm, produce a stable sort, which means that preserves the input order of equal elements in the sorted output. Mergesort is a divide and conquer algorithm that was invented by John von Neumann in 1945.

Function call

```
i64 merge_sort(int data[], int size)
{
    for(int i=0; i<size; i++) // Initialize
    {
        m_array[i] = m_aux[i] = data[i];
    }
    i64 movs = merge(m_array,m_aux,0,size-1);
    return movs;
}
```

Implementation

```
i64 merge(int v[], int va[], int L, int R) {
    i64 cnt = 0LL;
    if(L<R) {
        int mid = (L+R)/2;
        cnt += merge(va, v, L, mid);
        cnt += merge(va, v, mid+1, R);
        int i=L, j=mid+1, k=L;
        while( i<=mid && j<=R ) {
            if(va[i]<=va[j])
                v[k++]=va[i++];
            else {
                v[k++]=va[j++];
                cnt+=(mid+1) - i;
        } }
        while(i<=mid) v[k++]=va[i++];
        while(j<= R ) v[k++]=va[j++];
    }
    return cnt;
}
```

Heap Sort

Is a comparison-based sorting algorithm. It divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the maximum.

Function call

```
void heap_sort(int data[], int size) {
    heap_init();
    for(int i=1; i<=size; i++){
        heap_insert(h_aux, data[i-1]);
    }
    for(int i = 0; i < size; i++){
        heap_delete(h_aux, h_array[i]);
    }
}
```

Insertion

Insert function

```
void heap_insert(int heap[], int val) {
    int parent=0, node=++h_size;
    heap[node] = val;
    while(!is_root(node)) {
        parent = get_parent(node);
        if(heap[node] >= heap[parent])
            break;
        swap(heap[node], heap[parent]);
        node = parent;
        }
}
```

Deletion

Delete function

```
void heap_delete(int heap[], int &val) {
    val = heap[h_root]; h_size--;
    if(h_size >= h_root)
    {   // Set the last element
        heap[h_root] = heap[h_size+1];
        int node, small=h_root;
        do{     node = small;
            int left = node << 1;
            int right = left + 1;
            if(left<=h_size && heap[left]<heap[small])
                small = left;
            if(right<=h_size && heap[right]<heap[small])
                small = right;
            swap(heap[node],heap[small]);
        }while(small != node);
    } }
```

Quick Sort

Is an efficient sorting algorithm. Developed by Tony Hoare in 1959. When implemented well, it can be about two or three times faster than its main competitors, merge sort and heapsort.
Quicksort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation is defined. In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved. Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting.

Function call

```
void q_sort(int A[], int p, int r) {
    if(p < r) {
        int q = partition(A,p,r,q);
        q_sort(A, p, q-1);
        q_sort(A, q+1, r);
    }
}
```

Input based Comparison

Input	Ordered	Ordered Inverse	Almost ordered	Random
Merge	7547	7578	9305	17387
Heap	30941	45921	28126	33257
QS Fixed	37705066	25637125	2545834	19879
QS Random	12464	13249	13927	22118

Table: Time spent in us

Conclusion

Conclusion

Based on data obtained, we can conclude the algorithm better fit our propose in time complexity is merge sort, also we could notice an improvement when instead of take the last element in partition section of quick sort we take a random pivot.