List of Algorithms

Opening Questions

1. Mergesort is $\Theta(O(n \lg n))$ runtime in best case, worst case and average case. How much memory is needed for Mergesort on input size n? $O(n)$
2. Mergesort does all the work of sorting items in the Merge function, after recursively splitting the collection down to the base case. Briefly explain the difference with Quicksort.All the work for Mergesort is being done after the recurrsive calls, where as Quicksort does the work before the recurrsive calls.

Quicksort

A recursive divide and conquer algorithm.

- base case: a list of length one element is sorted.
- Divide "Partition" array into 2 sub-arrays with small #'s in the beginning, large #'s in second and known index dividing themWhat defines small and large? The dividing index may not be the middle.
- Conquer - recursively sort each sub-array
- Combine - Nothing to do

1. Write pseudocode for Quicksort (similar to Mergesort).

Algorithm 1 Quicksort

```
1: function QSort(A, p, r)                    ▷ The initial call is QSort(A, 1, n)
2:     if p < r then
3:         q ← Partition(A, p, r)      ▷ Moves small to left and large to right, q is the seperator
4:         QSort(A, p, q-1)                  ▷ The value at A[q] is the pivot, q is the idnex
5:         QSort(A, q+1, r)
6:     end if
7: end function
```

Partition idea (you should be able to do this in place): pick the last element in the current array as the "pivot", the number used to decide large or small. Then make a single pass of the array to move the "small" numbers before the "large" numbers and keep the "large" numbers after the "small" numbers. Then put the "pivot" between the two subarrays and return the location of the pivot.

2. Write iterative pseudocode for Partition. How much memory is needed?

Algorithm 2 Partition: Worst case runtime $O(n)$

```
1: function PARTITION(A,p,r)
2:     pivot ← A[r]
3:     i ← p − 1
4:     for j=p to r-1 do
5:         if A[j] ≤ pivot then
6:             Swap A[i+1] with A[j]
7:             increment i
8:         end if
9:     end for
10:    Swap A[i+1] with A[r]
11:    return i+1
12: end function
```

Quicksort Runtime:

$$T(n) = O(n) + T(\text{small items}) + T(\text{large items})$$
$$T(1) = O(1) + (O \leftrightarrow n-1) + (n-1 \leftrightarrow O)$$

If All Items $<$ Pivot

$$\begin{aligned} T(n) &= O(n) + T(n-1) + T(0) \\ &= O(1) + T(1-1) + O(1) \\ &= O(1) + T(0) + O(1) \\ &= O(1) \end{aligned}$$

$$\begin{aligned} T(n) &= O(n) + T(n-1) \\ &= O(n) + O(n-1) + T(n-2) \\ &= O(n) + O(n-1) + O(n-2) + T(n-3) \\ &= O(n) + O(n-1) + O(n-2) + O(n-3) + \cdots + T(1) \\ &= O(n) + O(n-1) + O(n-2) + O(n-3) + \cdots + O(1) \\ &= O\left(n^2\right) \end{aligned}$$

If Pivot is Median

$$\begin{aligned} T(n) &= O(n) + 2T\left(\frac{n}{2}\right) \\ &\quad \text{(Master method: } a = 2, b = 2, f(n) = O(n)) \\ &= n^{\log_b(a)} \\ &= n^{\log_2(2)} \\ &= n^1 \\ &= O(n \lg n) \end{aligned}$$

What if pivot divides items $\frac{1}{10}$ on oneside and $\frac{9}{10}$ on other

$$\begin{aligned} T(n) &= T\left(\frac{n}{10}\right) + T\left(\frac{n}{10}\right) + O(n) \\ &\cdots \\ &= \log_{\frac{10}{9}}(n) \\ &= O(n \lg n) \end{aligned}$$

As long as the split is some sort of ratio, you get $O(n \lg n)$ as the runtime for quicksort. Only in the rare case of all the data being on one side of the partition that gives $O(n^2)$ as the partition. To fix this, pick 3 items randomly, chose the middle of those 3, swap that middle with the last index and you'll be guaranteed to have $O(n \lg n)$.

Algorithm 3 Partition Improved

```
1:  function PARTITIONI(A,p,r)
2:      b, c, d are random indexes from A.
3:      Swap A[r] with MEDIAN(A[b], A[c], A[d])
4:      pivot ← A[r]
5:      i ← p − 1
6:      for j=p to r-1 do
7:          if A[j] ≤ pivot then
8:              Swap A[i+1] with A[j]
9:              increment i
10:         end if
11:     end for
12:     Swap A[i+1] with A[r]
13:     return i+1
14: end function
```

3. Demonstrate Partition on this array.

4. What do you think the best possible outcome would be for a call to Partition, and why? What about worst possible outcome?

5. Write (and solve) recurrence relations for Quicksort in the best case partition and worst case partition.

6. What is there is a pretty bad, but not awful, partition at every call. Try always a 9 to 1 split from partition. Write and solve recurrence relation.

With all the other sorts we could describe a particular input order that would yield worst case run time.

7. How can we avoid a particular input order yielding worst case run time for quicksort?

Visual Sorting Software By A. Alegoz, previous CS430 student (1.8Mb zipped, Win only)http://www.cs.iit.edu/~cs430/IITSort.zip