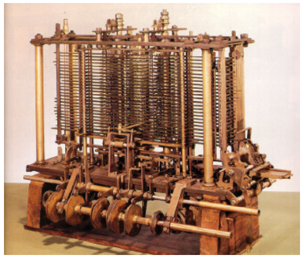
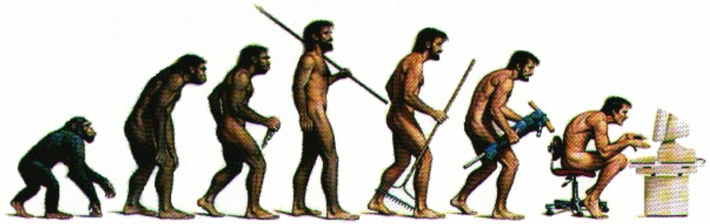




Data Structures and Algorithms II

CMPSC 130B

Divide and conquer

- A general paradigm for algorithm design
- Three-step process:
  - Divide the problem into smaller problems.
  - Conquer by solving these problems.
  - Combine these results together.
- How is it different from Greedy?

Divide and conquer examples

- Binary search
- MergeSort
- QuickSort
- Long multiplication
- Matrix multiplication
- Solving recurrences
- Selection in linear time
- Convex hull
- Closest pair of points

Binary search

- Let $T(n)$ denote the worst-case time to binary search in an array of length n
- Recurrence is $T(n) = T(n/2) + O(1)$
- $T(n) = ?$

MergeSort

- Merge-sort on an input sequence S with n elements consists of three steps:
  - Divide: partition S into two sequences S_1 and S_2 of about $n/2$ elements each
  - Recur: recursively sort S_1 and S_2
  - Conquer: merge S_1 and S_2 into a unique sorted sequence

Algorithm *mergeSort*(S, C)
  Input sequence S with n elements, comparator C
  Output sequence S sorted according to C
  if $S.size() > 1$
    $(S_1, S_2) \leftarrow partition(S, n/2)$
    $mergeSort(S_1, C)$
    $mergeSort(S_2, C)$
    $S \leftarrow merge(S_1, S_2)$

Analysis of MergeSort

- $T(n) = 1$ if $n = 1$

  $= 2\,T(n/2) + n$ otherwise.

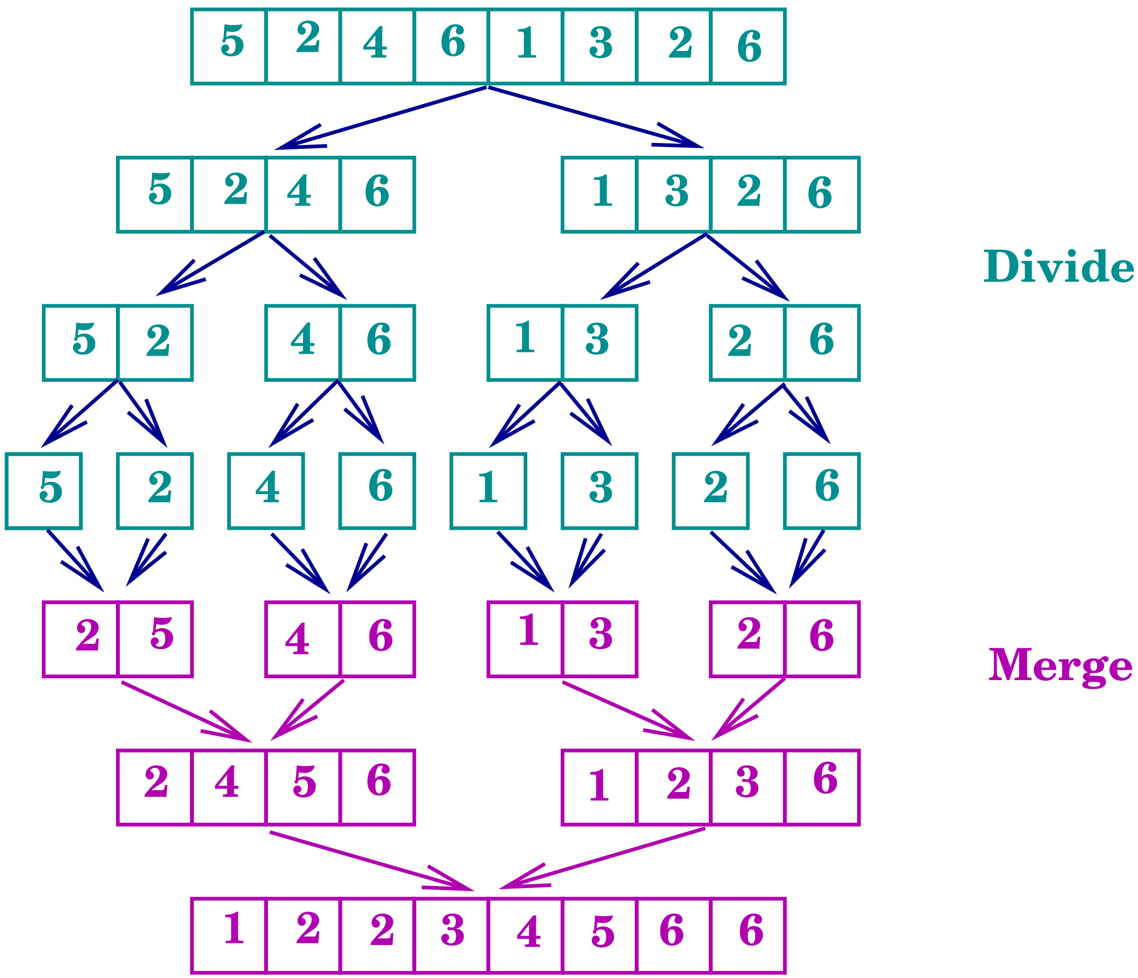
- Solve the recurrence by unraveling it:
  - $T(n) = 2\,(2\,T(n/4) + n/2) + n$

    $= 2^2\,T(n/2^2) + 2n$

    ....

    $= 2^i\,T(n/2^i) + i.n$
  - When $i = \log n$, $2^i = n$ and $n/2^i = 1$
  - $T(n) = n + n \log n$

MergeSort example



Quicksort

- Another divide-and-conquer algorithm
  - The array A[p..r] is *partitioned* into two subarrays A[p..q-1] and A[q+1..r], A[q] is the pivot.
    - Invariant:

      elements in A[p..q-1] <= A[q] <= elements in A[q+1..r]
  - The subarrays are recursively sorted by calls to quicksort
  - Unlike merge sort, no combining step: two subarrays form an already-sorted array

Quicksort code

```
Quicksort(A, p, r)
{
    if (p < r)
    {
        q = Partition(A, p, r);
        Quicksort(A, p, q-1);
        Quicksort(A, q+1, r);
    }
}
```

Partition operation

- Clearly, all the action takes place in the **partition()** function
  - Rearranges the subarray in place
  - Returns the index of the "pivot" element separating the two subarrays

Choosing pivot element

- First or last array element.
  - Good if array is random
  - Bad if array is partly sorted. Not recommended.
- Random pivot.
  - Generally works very well.
  - Recommended.
- Median of 3.
  - Pick 3 random elements and choose their median. Or, median of left, right, and middle.
  - E.g. in array (8, 1, 4, 9, 6, 3, 5, 2, 7, 0), left=8, right=0, center=6. So, the median is 6.

Partition details

- Partition(A, p, r):
  - Select an element A[k] in A[p..r] to act as the "pivot"
  - Grow two regions, A[p..i] and A[j..r] such that
    - all elements in A[p..i-1] $\leq$ pivot
    - pivot $\leq$ all elements in A[j+1..r]

p <= i <= j+1 <= r+1

  - i := p; j := r
  - Repeat
    - Increment i until A[i] > pivot or i = j+1
    - Decrement j until A[j] < pivot or i = j+1
    - If i < j

      Swap A[i] and A[j]; increment i; decrement j
  - Until i = j+1
  - If k < i

    Swap A[k] and A[i-1]; Return i-1

    Else

    Swap A[k] and A[j+1]; Return j+1

Notes

Analyzing Quicksort

- Suppose the pivot splits input into two subarrrays of sizes i and n-i-1. Assuming a linear time for partitioning and choosing the pivot,
  - $T(n) = T(i) + T(n-i-1) + cn$
  - $T(1) = 1, T(0) = 1$
- *Worst case?*
  - Partition is always unbalanced
- *Best case?*
  - Partition is balanced
- *Which is more likely?*
  - Poll

Analyzing Quicksort (worst case)

- Pivot is the smallest (or largest element).
- $T(n) = T(0) + T(n-1) + cn$
- Iterating,
  - $T(n-1) = T(n-2) + c(n-1)$
  - $T(n-2) = T(n-3) + c(n-2)$
  - ....
  - $T(2) \quad = T(1) + c.2$
- Thus, $T(n) = T(1) + c\,\Sigma\, i \; = O(n^2)$.

Analyzing Quicksort (best case)

- Pivot evenly splits the array every time during recursion.
- $T(n) = 2\, T(n/2) + cn$
- $T(n) = O(n \log n)$, using same logic as Merge sort
- Another method:
  - $T(n) = 2T(n/2) + cn$
  - $T(n)/n = T(n/2)/(n/2) + c$, divide both sides by n
  - $T(n/2)/(n/2) = T(n/4)/(n/4) + c$, repeating
  - $T(n/4)/(n/4) = T(n/8)/(n/8) + c$,
  - ....
  - $T(2)/2 = T(1)/1 + c$;.
  - Adding and telescoping:

    $T(n)/n = T(1)/1 + c \log n$.
  - So, $T(n)$ is $O(n \log n)$.

Notes

Average case analysis

What happens if we get sort-of-balanced partitions,
e.g., something like:

$$T(n) = T(9n/10) + T(n/10) + O(n) \ ?$$

Still get O(n log n)

Intuition: Can divide n by c > 1 only O(log n) times before getting 1.

$$\left.\begin{array}{c} n \\ \downarrow \\ n/c \\ \downarrow \\ n/c^2 \\ \downarrow \\ \vdots \\ \downarrow \\ 1 \end{array}\right\} \text{roughly, } \log_c n \text{ steps}$$

Intuition holds even if c is very close to 1, e.g., 100/99.

Analyzing Quicksort (expected case)

- Assume the pivot is chosen at random and that S is split into S1 and S2.
- The size of the S1 subproblem is i, for i = 0,1,...,n-1, with equal probability.
- Same for the size of S2.

Notes

Solving the recurrence

$$T(n) = cn + \frac{1}{n}\sum_{i=0}^{n-1} T(i) + T(n-i-1)$$

$$T(n) = cn + \frac{2}{n}\sum_{i=0}^{n-1} T(i)$$

$$\mathrm{n}T(n) = cn^2 + 2\sum_{i=0}^{n-1} T(i)$$

$$(n-1)T(n-1) = c(n-1)^2 + 2\sum_{i=0}^{n-2} T(i)$$

$$nT(n) - (n-1)T(n-1) = 2cn - c + 2T(n-1)$$

$$nT(n) = (n+1)T(n-1) + 2cn \; - c$$

$$nT(n) = (n+1)T(n-1) + 2cn$$

Solving (continued)

$$nT(n) = (n+1)T(n-1) + 2cn$$

Dividing by $n(n+1)$,

$$\frac{T(n)}{n+1} = \frac{2c}{n+1} + \frac{T(n-1)}{n}$$

$$= \frac{2c}{n+1} + \frac{2c}{n} + \frac{T(n-2)}{n-1}$$

$$= \cdots$$

$$= 2c\sum_{k=3}^{n+1}\frac{1}{k} + \frac{T(1)}{2}$$

$$= 2c\left(H_{n+1} - \frac{1}{1} - \frac{1}{2}\right) + \frac{T(1)}{2}$$

Since, H_n is $O(\log n)$,

$T(n)$ is $O(n \log n)$

Notes