CS472 Module 4 Part D - Divide and Conquer - Characteristics and Mergesort

Athens State University

February 5, 2016

Outline

Contents

1 What is divide-and-conquer?

Let's reconsider Quicksort

- Let's pivot an array on the array's first element

p	A[i]<=p	A[i]>=p

- Now exchange the pivot with the last element in the first partition
  - We have placed the pivot in its correct spot
- Sort the two partitions recursively

The divide-and-conquer meta-heuristic

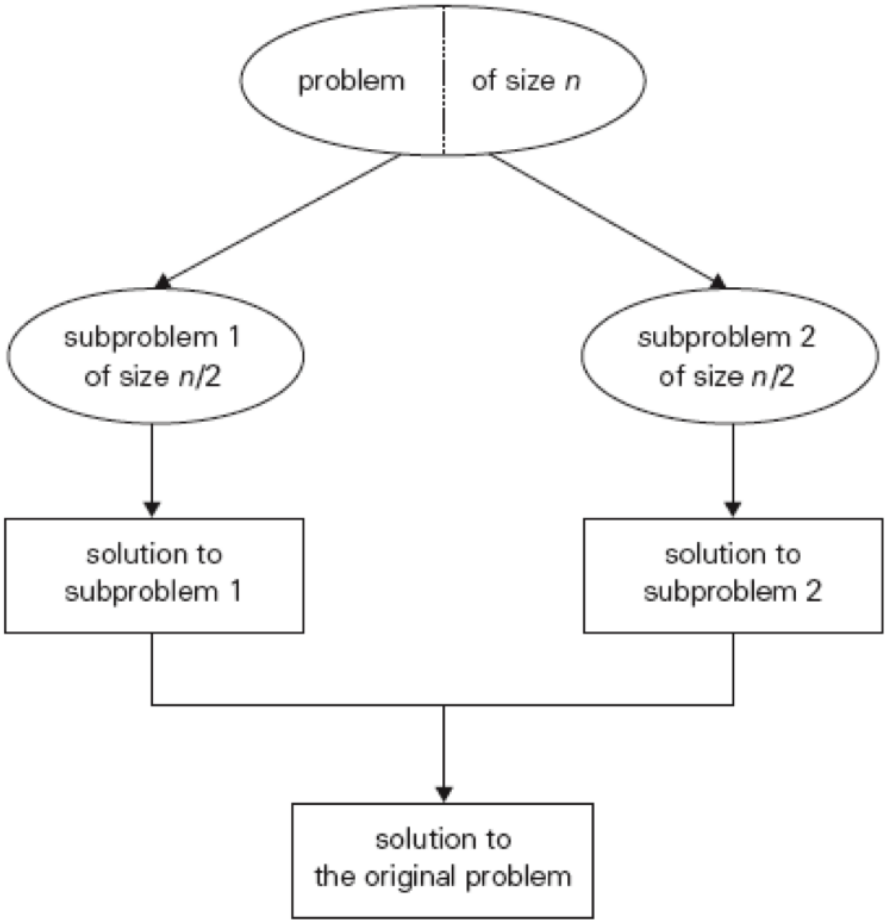

FIGURE 5.1 Divide-and-conquer technique (typical case).

Revisiting the analysis of Quicksort

- In the best case, luck smiles upon us and we end up selecting the key for the median value in the array
  - So, half of the keys go left and half the keys go right
- Plus, half of the time, we will choose a pivot that is going to be in the center half of range of values being sorted
- Which would generate a recurrence relation that looks like:

$$C(1) = 0 \tag{1}$$
$$C(n) = 2C\left(\frac{n}{2}\right) + n \tag{2}$$

- So, how to go about finding a general form of $C(n)$?
- Fortunately, we have a way to avoid the hard work…

Revisiting the analysis of Quicksort: The Master Theorem

- The recurrence relation $C(n)$ is an example of a *general divide-and-conquer relation*

$$T(n) = aT(N/b) + f(n)$$

- In this case, we are assuming that we divide a problem n into a collection of a sub-problems of size n/b.
  - Assume n is a power of b to keep things simple.
- The function $f(n)$ accounts for the time required for dividing an instance of size n into instances of size n/b and combining their solutions

Revisiting the analysis of Quicksort: The Master Theorem

Master Theorem: If $f(n) \in \Theta(n^d)$ with $d \geq 0$ in a general divide-and-conquer relation, then

$$T(n) \in \left\{ \begin{array}{ll} \Theta(n^d) & \text{if} a < b^d \\ \Theta(n^d * log(n)) & \text{if} a = b^d \\ \Theta(n^{log_b(a)}) & \text{if} a > b^d \end{array} \right\}$$

Revisiting the analysis of Quicksort: Applying the Master Theorem

- So, for Quicksort, we have the recurrence relation

$$C(1) = 0 \tag{3}$$

$$C(n) = 2C\left(\frac{n}{2}\right) + n \tag{4}$$

- Thus, we have $a = 2$, $b = 2$, and $d = 1$ for this version of a divide-and-conquer recurrence
- So, from the Master Theorem, we can quickly see that Quicksort is $\Theta(n * log(n)$
- That's a lot simpler than solving the recurrence by exhaustion, isn't it?

2 Mergesort

Mergesort: Overview

- Quicksort divides the array to be sorted based on the values in the array
- Suppose we divide the array according to the position of the elements in array and then sorted the subarrays
- So, if we're sorting an array $A[0..(n-1)]$, divide the array into as close to equal halves as possible and copy the two parts into new arrays B and C
- Recursively sort the two new arrays

Mergesort: Overview

- Now that we have the arrays B and C sorted, we need to merge the result back into A
- Repeat the following until no elements remain in one of the arrays
  - Compare the first elements in remaining unprocessed portions of B and C
  - Copy the smaller of the two into A while incriminating the index indicating the unprocessed portion of that array
- Once we finish processing one of the arrays, copy the remaining unprocessed elements from the other array into A

Mergesort: Example

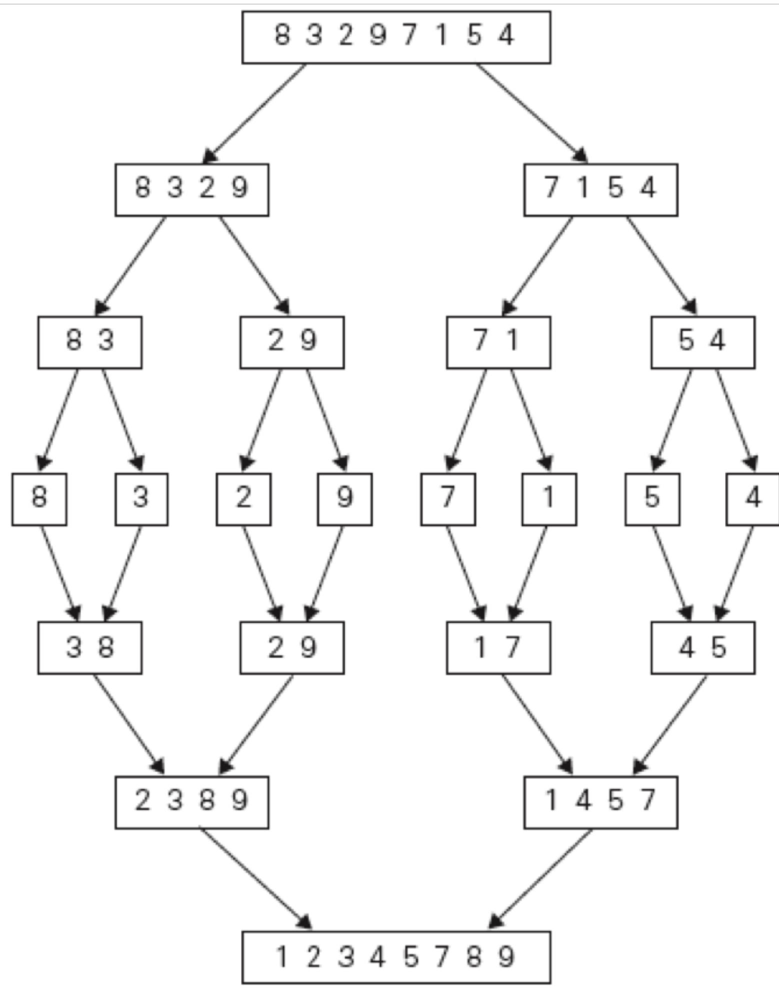

FIGURE 5.2 Example of mergesort operation.

Mergesort: The Algorithm

Algorithm 1: Mergesort

```
Input: An array A[0..(n-1)] of orderable elements
Output: The array A[0..(n-1)] sorted in non-decreasing order
if n > 1 then
    bp ← Floor (n/2) - 1;
    copy A[0..(bp -1)] to B[0..(bp-1)];
    copy A[(bp-1)..(n-1)] to C[0..bp];
    Mergesort (B);
    Mergesort (C);
    Merge (B,C,A);
```

Mergesort: The Merge Process

Algorithm 2: Merge: merge two sorted arrays into one sorted array

```
Input: Sorted arrays B[0..(p-1)] and C[0..(q-1)]
Output: Sorted array A[0..(p+q-1)] of the elements of B and C
i ← j ← k ← 0;
while i < p and j < q do
    if B[i] ≤ C[j] then
        A[k] ← B[i];
        i ← i+1;
    else
        A[k] ← C[j];
        j ← j+1;
    k ← k + 1;
if i = p then
    copy C[j..(q-1)] to A[k..(p+q-1)];
else
    copy B[i..(p-1)] to A[k..(p+q-1)];
```

Mergesort: Analysis

- Let's keep things simple by assuming that size n of the array to be sorted is a power of 2
- So, we have the following recurrence relation (with base case $C(1) = 0$):

$$C(n) = 2C(n/2) + C_{\text{merge}}(n)$$

- Per the Master Theorem, we have efficiency of $\Theta(n * log(n))$
- And the worst case is $\Theta(n * log(n))$ as well
- Problem (of sorts) is that the algorithm doesn't work in-place in memory and has $\Theta(n)$ space requirements

3 Key Points

Key Points

- Nature of divide and conquer algorithms
- General divide-and-conquer recurrences and the Master Theorem
- Mergesort
  - Algorithm design
  - Analysis