KALAMAZOO COLLEGE

Introduction

COMP215: Design & Analysis of Algorithms

KALAMAZOO COLLEGE

Today

- Merge Sort: The Algorithm
- Merge Sort: The Analysis
- Guiding Principles for the Analysis of Algorithms

MergeSort: The Algorithm

- Why begin with MergeSort?
  - Oldie but a goodie, it is the standard sorting algorithm in a number of programming libraries.
  - Canonical divide-and-conquer algorithm.
  - Our running time analysis of MergeSort exposes a number of more general guiding principles.
  - Warm-up for the master method.

6 5 3 1 8 7 2 4

MergeSort: The Algorithm

- Sorting:

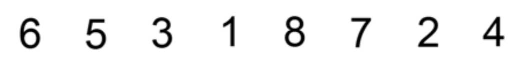
> Problem: Sorting
>
> Input: An array of n numbers, in arbitrary order.
>
> Output: An array of the same numbers, sorted from smallest to largest.

Some Algorithms:

- Insertion Sort
- Selection Sort
- Bubble Sort

MergeSort: The Algorithm

– Insertion Sort

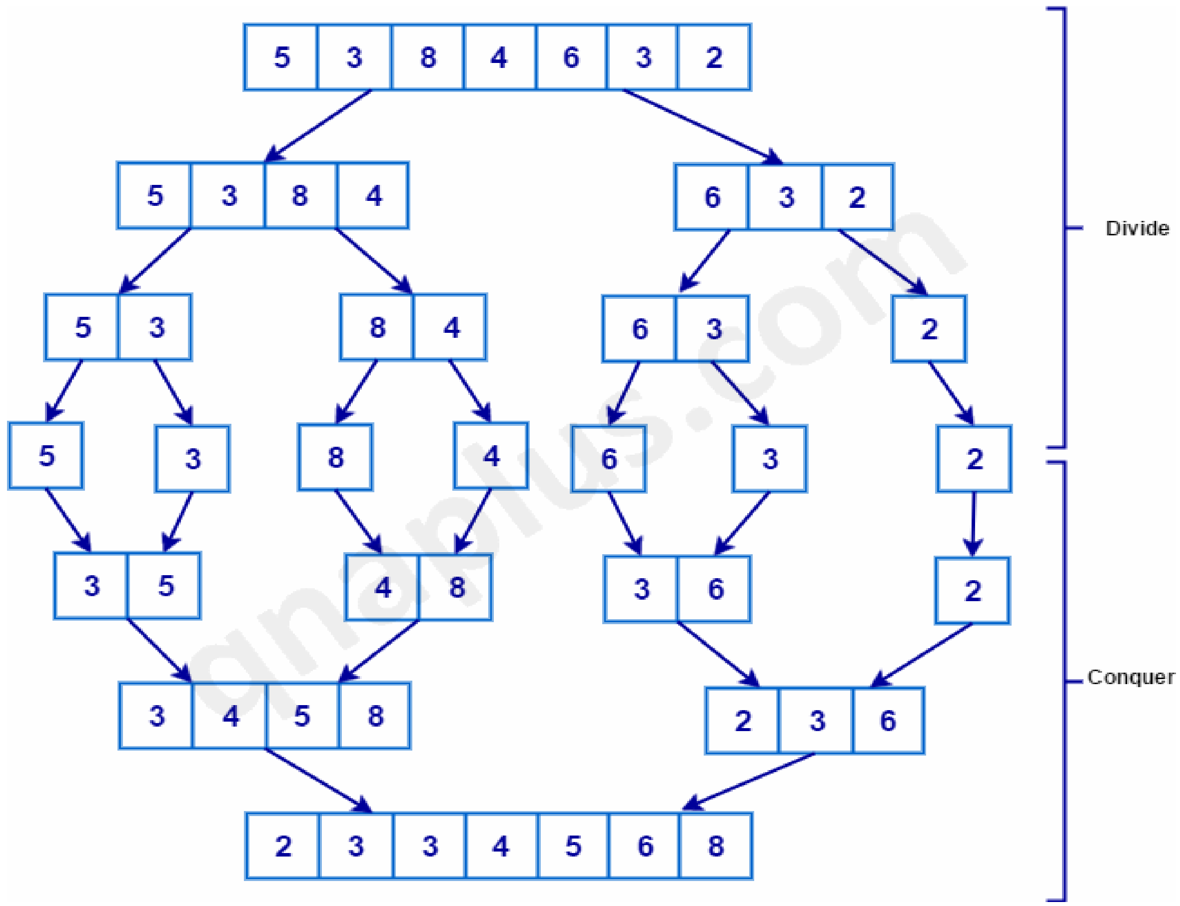


– Selection Sort

– Bubble Sort

MergeSort: The Algorithm

- Example:



MergeSort: The Algorithm

`MergeSort`

Input: array A of n distinct integers.
Output: array with the same integers, sorted from smallest to largest.

`// ignoring base cases`
C := recursively sort first half of A
D := recursively sort second half of A
return `Merge` (C, D)

`Merge`

Input: sorted arrays C and D (length $n/2$ each).
Output: sorted array B (length n).
Simplifying assumption: n is even.

$i := 1$
$j := 1$
for $k := 1$ to n **do**
  if $C[i] < D[j]$ **then**
    $B[k] := C[i]$ `// populate output array`
    $i := i + 1$ `// increment` i
  else `//` $D[j] < C[i]$
    $B[k] := D[j]$
    $j := j + 1$

MergeSort: The Analysis

- Running Time of Merge:

Lemma 1.1 (Running Time of Merge) For every pair of sorted input arrays C, D of length n/2, the Merge subroutine performs at most 6n operations.

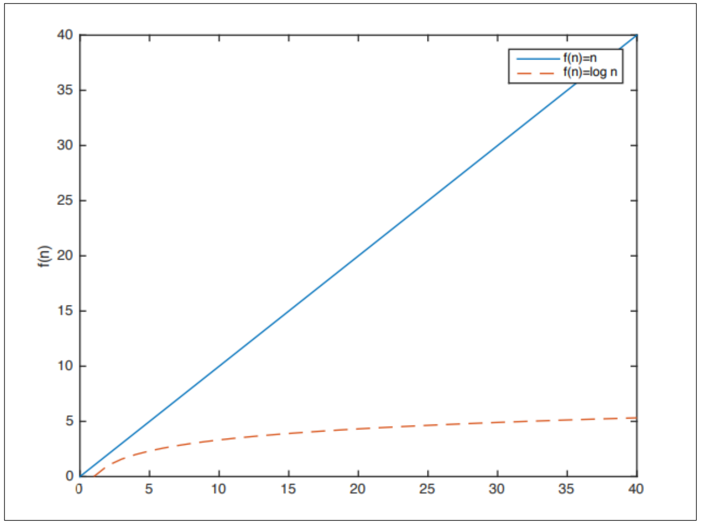
MergeSort: The Analysis

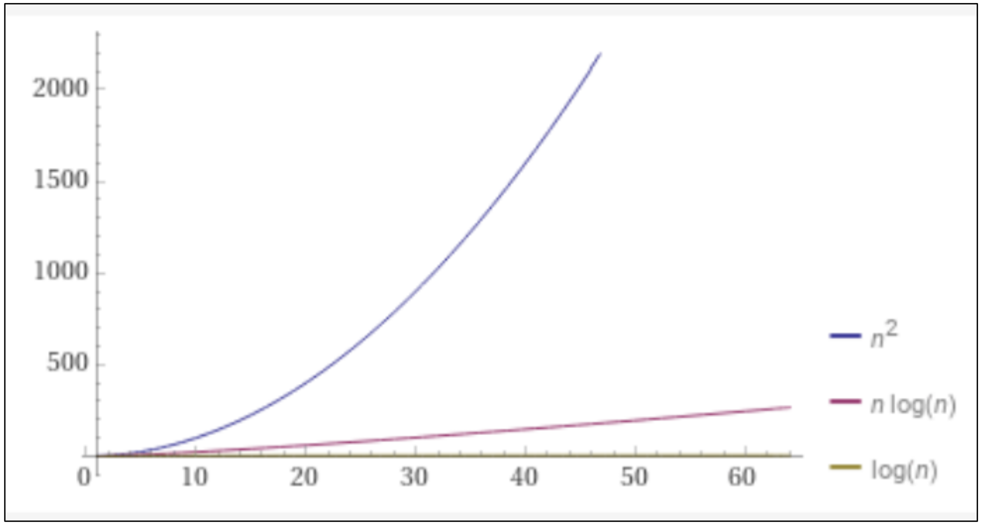
Theorem 1.2:

(Running Time of MergeSort) For every input array of length n >=1, the MergeSort algorithm performs at most :

$$6n \log_2 n + 6n$$

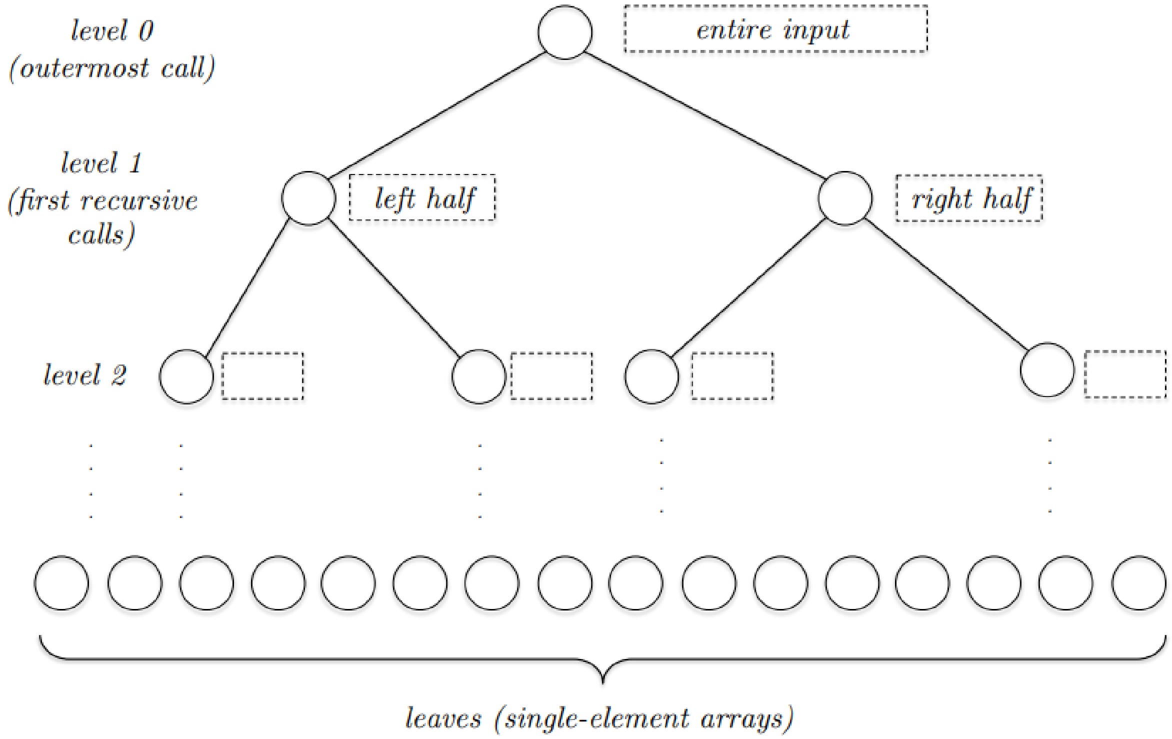
MergeSort: The Analysis





MergeSort: The Analysis

Proof: Running Time of MergeSort) For every input array of length n >=1, the MergeSort algorithm performs at most :**$6n \log_2 n + 6n$**



Roughly how many levels does this recursion tree have, as a function of the length n of the input array?

MergeSort: The Analysis

Quiz 1.2

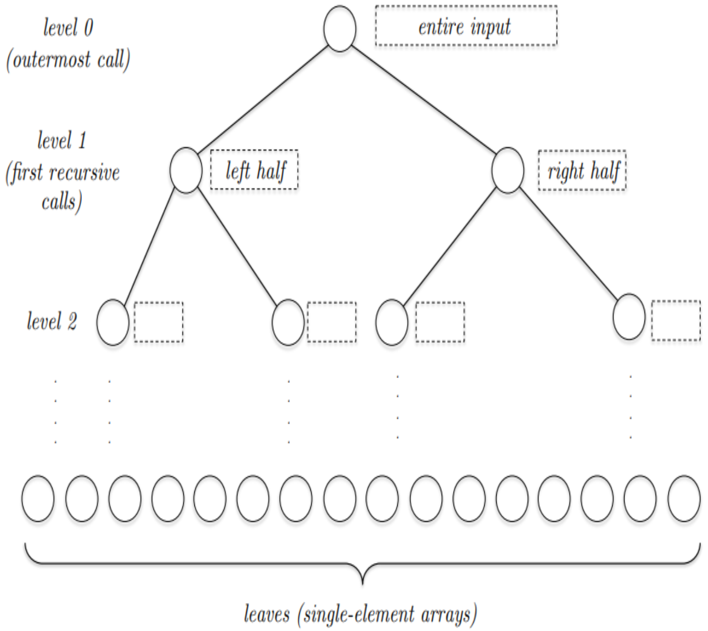
What is the pattern? Fill in the blanks in the following statement: at each level $j = 0, 1, 2, \ldots$ of the recursion tree, there are [blank] subproblems, each operating on a subarray of length [blank].

a) 2^j and 2^j, respectively

b) $n/2^j$ and $n/2^j$, respectively

c) 2^j and $n/2^j$, respectively

d) $n/2^j$ and 2^j, respectively



The total work done by level-j recursive call:
# of level-j subproblems * work per level-j subproblem
$= 2^j * 6n/2^j$
$=6n$

MergeSort: The Analysis

Using our bound of 6n operations per level, we can bound the total number of operations by
number of levels * work per level
$= (\log_2 n+1) *6n$
$= 6n \log_2 n + 6n$

Guiding Principles for the Analysis of Algorithms

- Principle #1: Worst-Case Analysis
  - This type of analysis is called worst-case analysis, since it gives a running time bound that is valid even for the "worst" inputs.
  - Worst-case analysis, in which you make absolutely no assumptions about the input, is particularly appropriate for general purpose subroutines designed to work well across a range of application domains
- Principle #2: Big-Picture Analysis
  - This principle states that we should not worry unduly about small constant factors or lower-order terms in running time bounds
- Principle #3: Asymptotic Analysis
  - Focus on the rate of growth of an algorithm's running time, as the input size n grows large.
- What Is a "Fast" Algorithm?
  - A "fast algorithm" is an algorithm whose worst-case running time grows slowly with the input size.