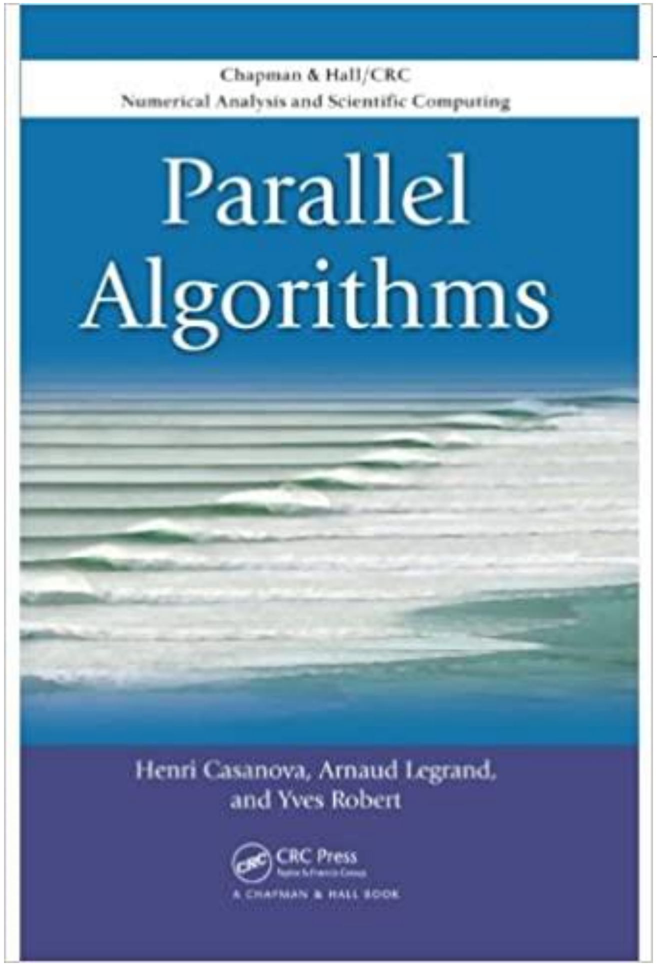
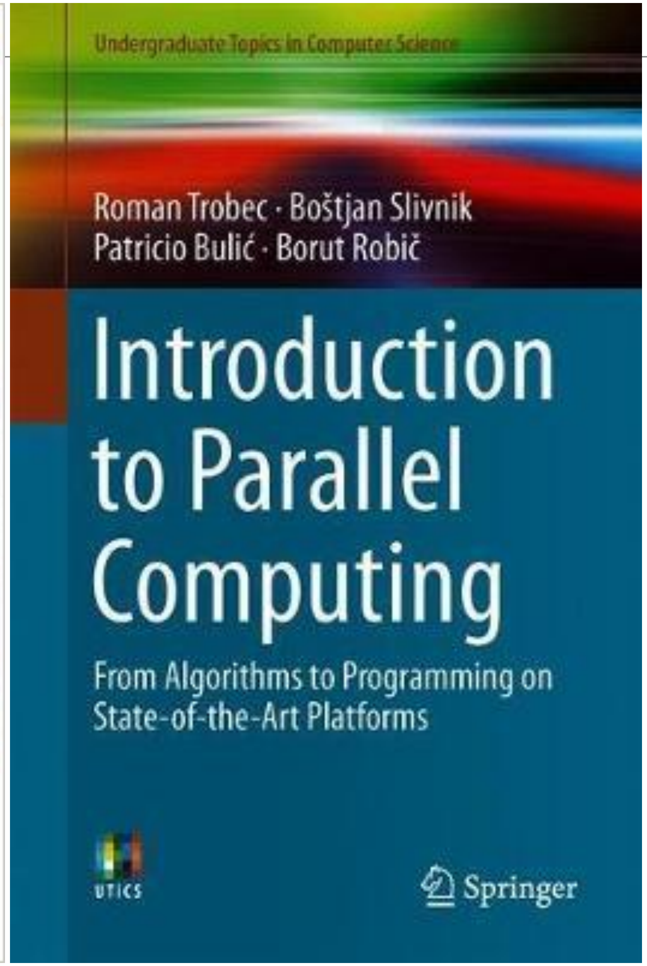
Parallel Programming

Lec 4

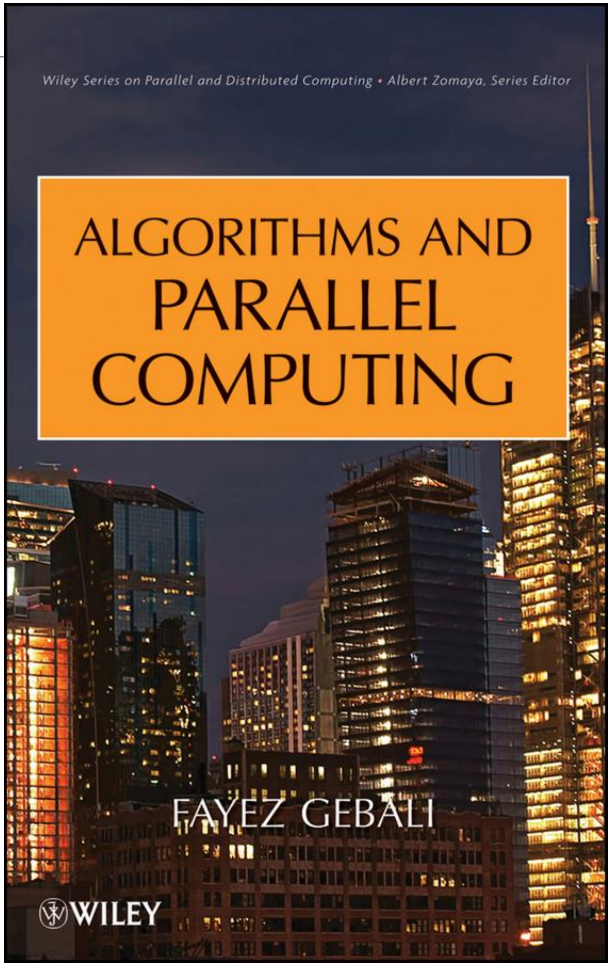
Books



Chapman & Hall/CRC
Numerical Analysis and Scientific Computing
Parallel Algorithms
Henri Casanova, Arnaud Legrand, and Yves Robert
CRC Press
A CHAPMAN & HALL BOOK



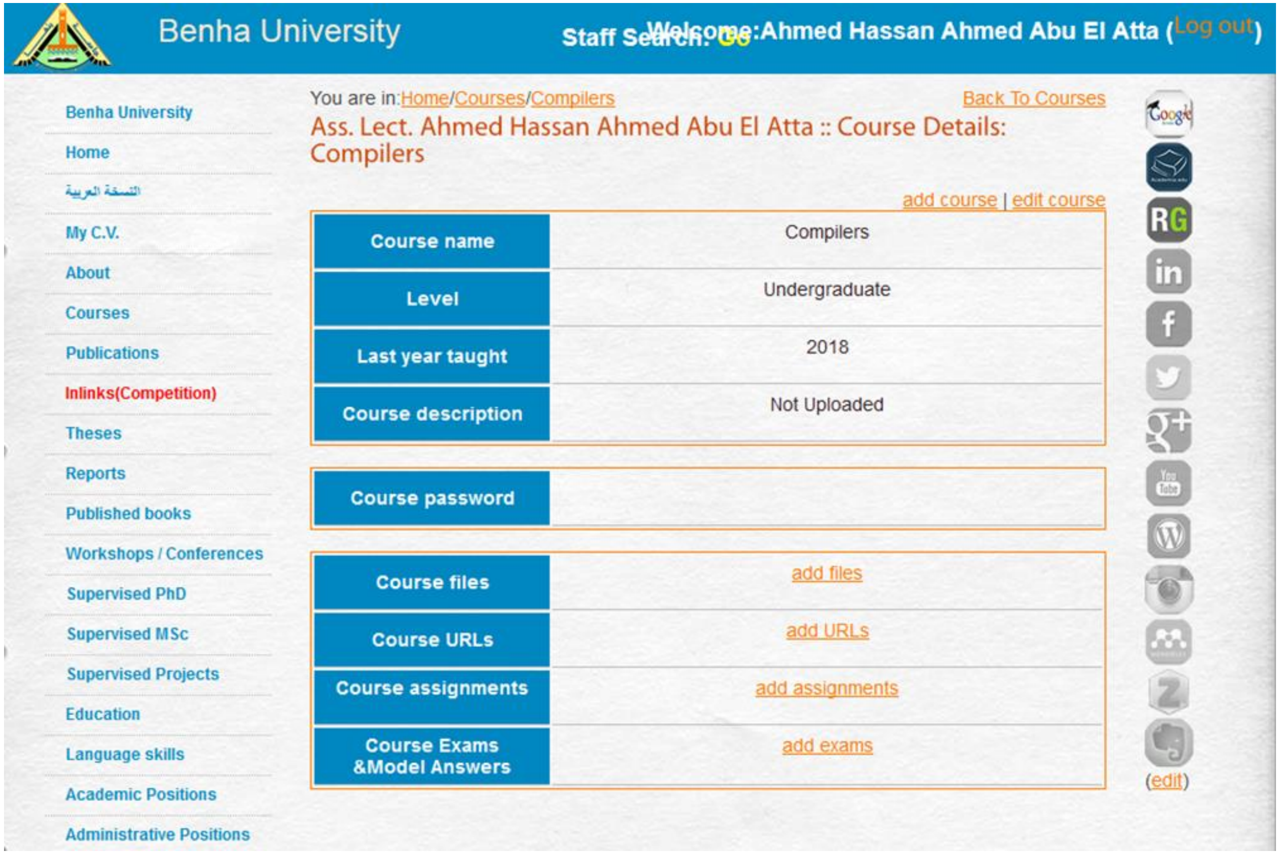
Undergraduate Topics in Computer Science
Roman Trobec · Boštjan Slivnik
Patricio Bulić · Borut Robič
Introduction to Parallel Computing
From Algorithms to Programming on State-of-the-Art Platforms
UTICS
Springer



Wiley Series on Parallel and Distributed Computing • Albert Zomaya, Series Editor
ALGORITHMS AND PARALLEL COMPUTING
FAYEZ GEBALI
WILEY

PowerPoint

http://www.bu.edu.eg/staff/ahmedaboalatah14-courses/14779



Sorting

Sorting is the process of ordering a set of values into ascending (lowest to highest) or descending (highest to lowest) order.

Sorting is used in compilers, editors, memory management and process management and is one of the most important operations performed in computers.

There are several sequential sorting algorithms, such as ShellSort, MergeSort, QuickSort, TreeSort, HeapSort and BingoSort, which are comparision-based sorting algorithms.

Two Approaches to Sorting

Sorting by Merging.

In this method, the sequence to be sorted is divided into two subsequences of equal length.

Each of the two subsequences is now sorted recursively.

Finally, the two sorted subsequences are merged into one sorted sequence, thus providing the answer to the original problem.

Two Approaches to Sorting

Sorting by Splitting.

In this method, the sequence to be sorted is divided into two subsequences of equal length such that each element of the first subsequence is smaller than or equal to each element of the second subsequence.

This splitting operation is then applied to each of the two subsequences recursively.

When the recursion terminates, the sequence is sorted.

Compare-and-Exchange

We assume that the elements to be sorted are integer numbers and are resident in an array, and for simplicity we take the number of elements to be a power of 2(i.e. 0, 2, 4, 8, 16, 32, …, 2^n, …).

As is usual, we suppose that each element of the array to be sorted has a key which governs the sorting process.

If the keys of the items to be sorted are in a vector Key[1 ... n], the operation of compare-and-exchange can be defined as follows:

```
Compare-and-Exchange(i,j)
if (Key[i] > Key[j] ) { // sorting in increasing order i<j
temp = Key[i] ;
Key[i] = Key[j];
Key[j] = temp;
}
```

Parallel Compare-and-Exchange

This indicates that two compare-and-exchange operations can be performed simultaneously if and only if they operate on disjoint entries of the vector Key.

Version 1 – P_1 sends A to P_2, which then compares A and B and sends back to P_1 the min(A,B).

Version 2 – P_1 sends A to P_2 and P_2 sends B to P_1, then both perform comparisons and P_1 keeps the min(A,B) and P_2 keeps the max(A,B).

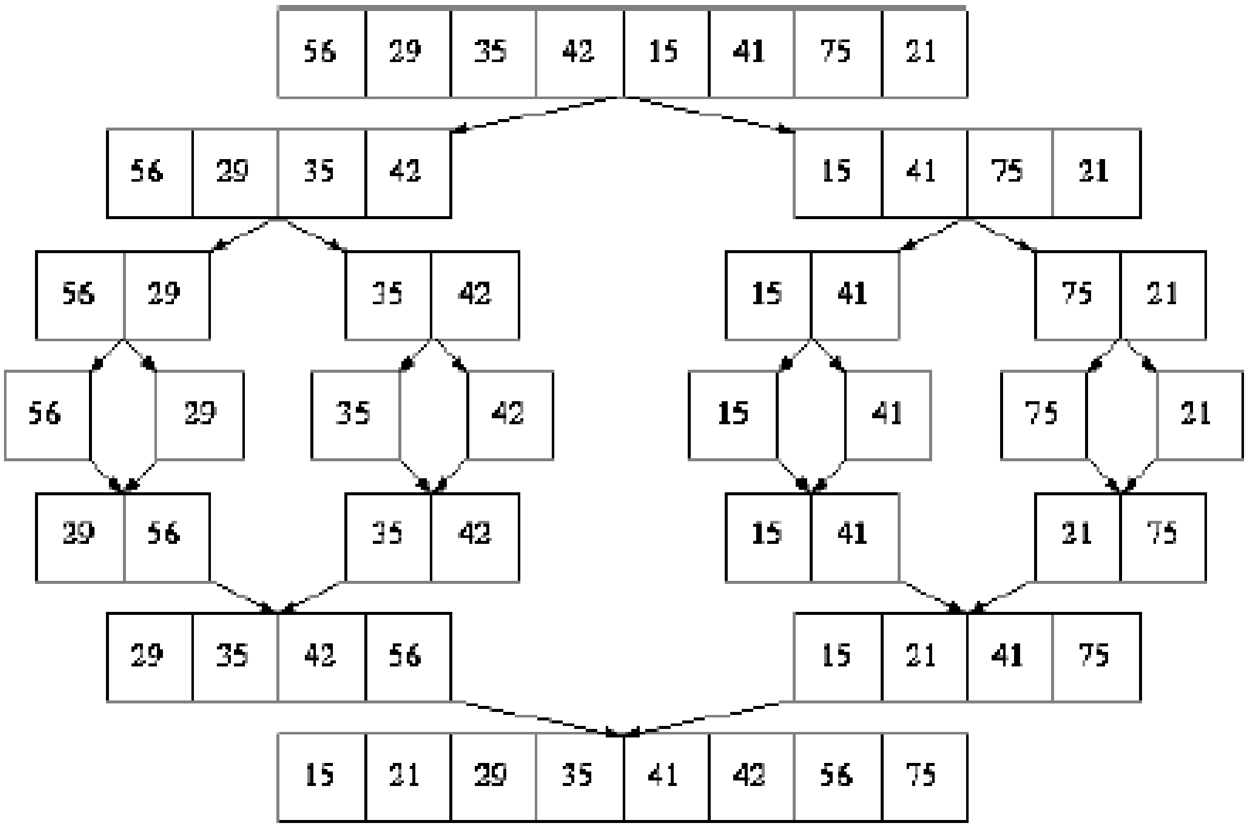
Mergesort

Mergesort is a classical sorting algorithm using a divide-and-conquer approach.

The initial unsorted list is first divided in half, each half sub-list is then applied the same division method until individual elements are obtained.

Pairs of adjacent elements/sub-lists are then merged into sorted sub-lists until the one fully merged and sorted list is obtained.

Mergesort


56 29 35 42 15 41 75 21
56 29 35 42
15 41 75 21
56 29
35 42
15 41
75 21
56
29
35
42
15
41
75
21
29 56
35 42
15 41
21 75
29 35 42 56
15 21 41 75
15 21 29 35 41 42 56 75

Mergesort

Computations only occur when merging the sub-lists.

In the worst case, it takes $2s - 1$ steps to merge two sorted sub-lists of size s. If we have $m = n/s$ sorted sub-lists in a merging step, it takes:

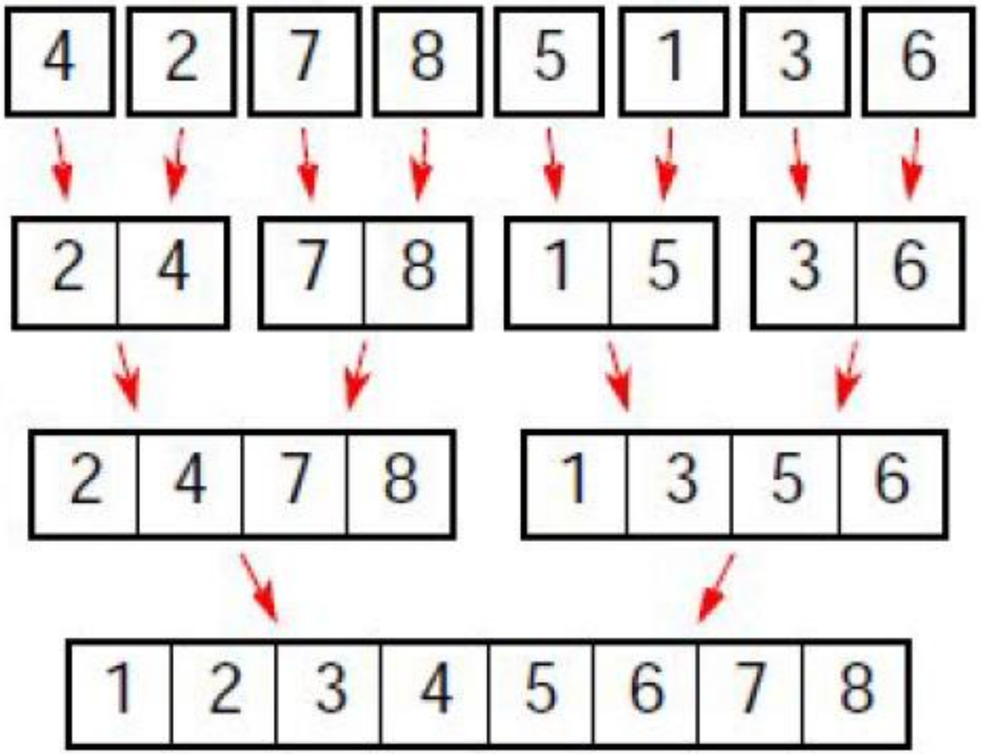
$$\frac{m}{2}(2s - 1) = ms - \frac{m}{2} = n - \frac{m}{2}$$

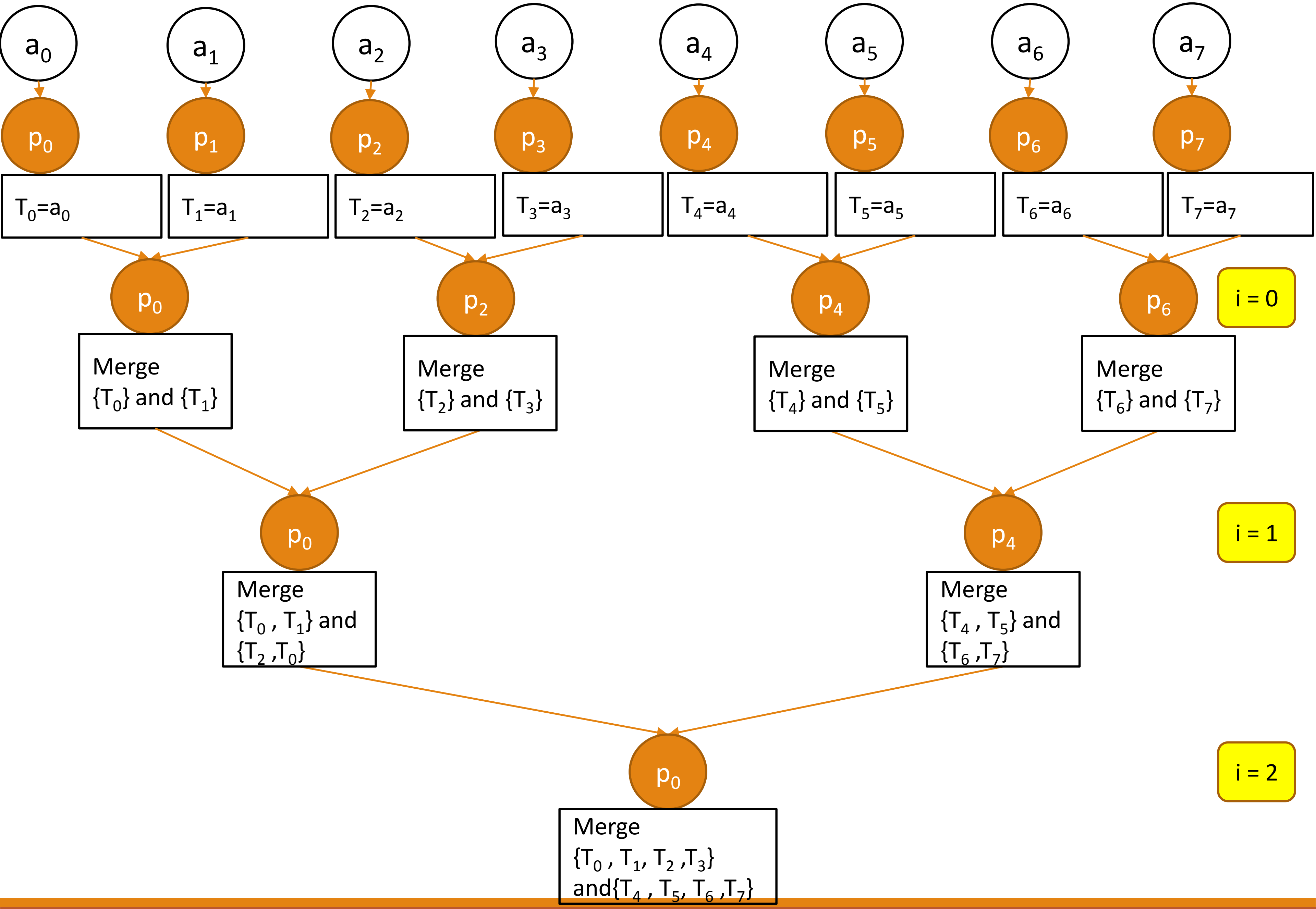
steps to merge all sublists (two by two).

Since in total there are log(n) merging steps, this corresponds to a time complexity of O(n log(n)).

Parallel Mergesort

The idea is to take advantage of the tree structure of the algorithm to assign work to processes.





a0
a1
a2
a3
a4
a5
a6
a7
p0
p1
p2
p3
p4
p5
p6
p7
T0=a0
T1=a1
T2=a2
T3=a3
T4=a4
T5=a5
T6=a6
T7=a7
p0
p2
p4
p6
i = 0
Merge {T0} and {T1}
Merge {T2} and {T3}
Merge {T4} and {T5}
Merge {T6} and {T7}
p0
p4
i = 1
Merge {T0 , T1} and {T2 ,T0}
Merge {T4 , T5} and {T6 ,T7}
p0
i = 2
Merge {T0 , T1, T2 ,T3} and{T4 , T5, T6 ,T7}

merge(Start, Mid, End)

```
i = k = Start, j = Mid + 1

While(i ≤ Mid and j ≤ end)
    If (T[i] ≤ T[j])
        A[k++] = T[i++]
    Else
        A[k++] = T[j++]

while(i ≤ Mid)
        A[k++] = T[i++]

while(j ≤ end)
        A[k++] = T[j++]
```

ParallelMergeSort(A, n)

For $i = 0$ to $i<\log(n)$ do

For $j = 0$ to $j <n$ do in parallel

$T[j] = A[j]$

For $j = 0$ to $j <n/(2^{(i+1)})$ do in parallel

merge($j * 2^{(i+1)}$, $(2j+1)*2^{i}-1$, $(j+1) * 2^{(i+1)}-1$)

Parallel Mergesort

If we ignore communication time, computations still only occur when

merging the sub-lists.

$$\sum(1 + 2\ + 4\ + 8 + 16 + \cdots + n) = \sum \frac{n}{1} + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{n}$$

$$= \sum_{0}^{logn} \frac{n}{2^i}$$

a_0 = n , r = ½

then the summation = a_0 /(1-r) = n/(1-1/2)) = n/(1/2) = 2n

It takes 2n steps to obtain the final sorted list in a parallel implementation, which corresponds to a time complexity of O(n)