Lectures 1-2.
Introduction - Insertion Sort - Merge Sort

Introduction to Algorithms

Da Nang University of Science and Technology

Dang Thien Binh

dtbinh@dut.udn.vn

Copyright 2000-2022 IT Faculty

Algorithm

- A well-defined computational procedure that transforms the input to the output
- Describes a specific computational procedure for achieving the desired input/output relationship
- ***An instance*** of a problem is all the inputs needed to compute a solution to the problem
- A ***correct algorithm***
  - *halts* with the *correct output* for every input instance
  - is said to *solve the problem*

Algorithm

- ***Example***: Sorting
  - ***Input***: A sequence of *n* numbers $<a_1, a_2, \ldots, a_n>$
  - ***Output***: A permutation (reordering) $<b_1, b_2, \ldots, b_n>$ of the input sequence such that $b_1 \leq b_2 \leq \cdots \leq b_n$
  - ***Instance***: <6, 4, 3, 7, 1, 4>

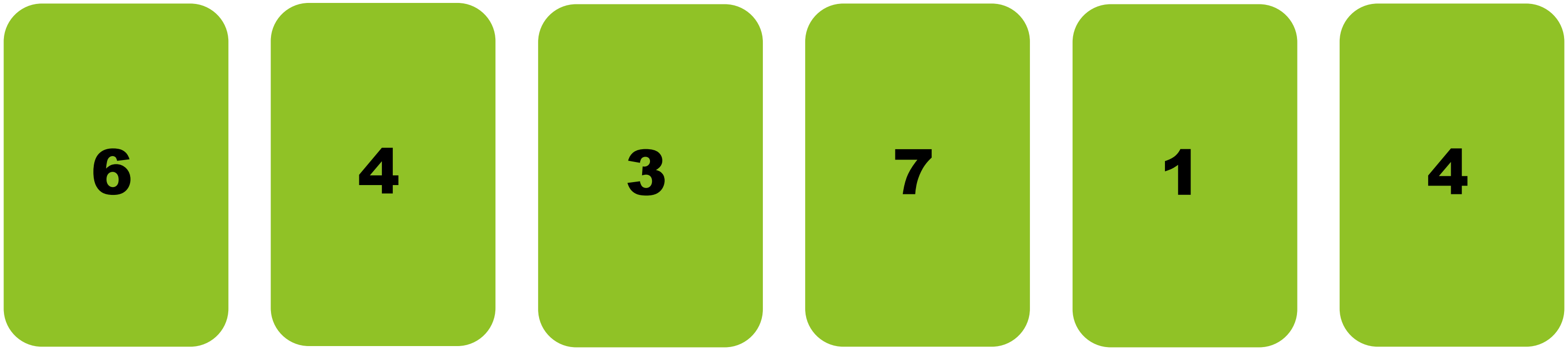
Algorithm

- ***Insertion Sort***
  - In-place sorting: Uses only a fixed amount of storage beyond that needed for the data
- ***Example***:

```
6 4 3 7 1 4          4 6 3 7 1 4
^ *                  ^   *
3 4 6 7 1 4          3 4 6 7 1 4
      *              ^       *
1 3 4 6 7 4          1 3 4 4 6 7
      ^   *
```

Algorithm

- ***Example***: 6 4 3 7 1 4



Algorithm

- ***Pseudocode***:

```
INSERTION-SORT(A)   /* A is an array of numbers */
for j ← 2 to length[A]
        key ← A[j]
        /* insert A[j] into the sorted sequence A[1 .. j-1] */
        i ← j - 1
        while i > 0 and A[i] > key
                A[i+1] ← A[i]
                i ← i - 1
        A[i+1] ← key
```

Algorithm

```cpp
#include <cstdlib>
#include <iostream>

using namespace std;

//member function
void insertion_sort(int arr[], int length);
void print_array(int array[],int size);

int main() {

int array[5]= {5,4,3,2,1};
insertion_sort(array,5);
print_array(arr,5);

 return 0;
}//end of main
```

```cpp
void insertion_sort(int arr[], int length) {
 int i, j ,tmp;
 for (i = 1; i < length; i++) {
     j = i;
     while (j > 0 && arr[j - 1] > arr[j]) {
         tmp = arr[j];
         arr[j] = arr[j - 1];
         arr[j - 1] = tmp;
         j--;
     }//end of while loop
 }//end of for loop
}//end of insertion_sort.
```

Algorithm

- ***Example:***
  - 5 2 4 6 1 3

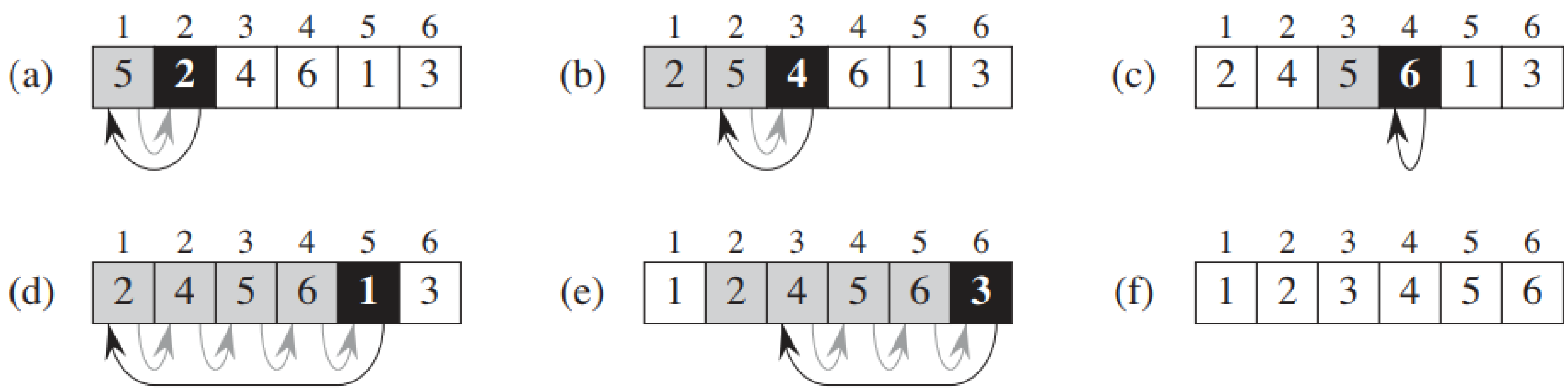
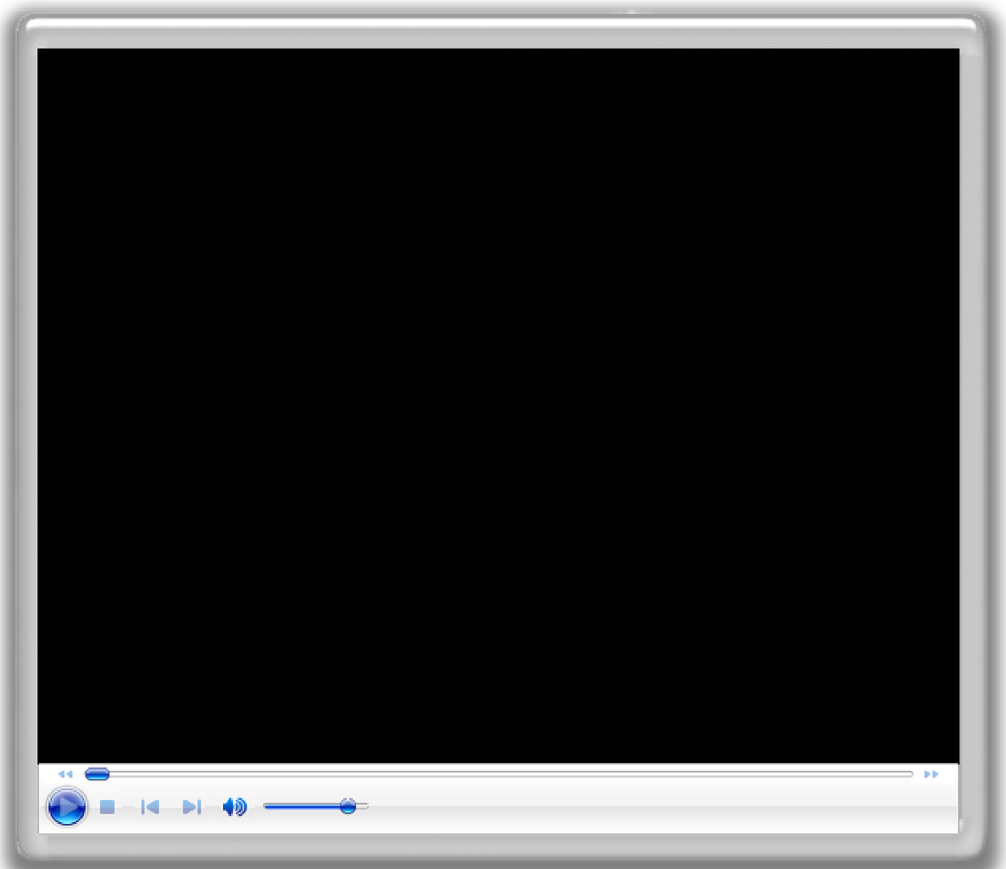


Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key moves to in line 8. **(f)** The final sorted array.

Insertion Sort Algorithm

- Video Content
  - An illustration of Insertion Sort with Romanian folk dance.

Insertion Sort Algorithm



Analyzing Algorithm

- Predicting the resources, such as memory, bandwidth, logic gates, or *running time*
- Assumed implementation model
  - *Random-access machine (RAM)*
- ***Running time***: f(input size)
- ***Input size***:
  - Sorting: number of items to be sorted.
  - Multiplication: number of bits.
  - Graphs: numbers of vertices and edges.

Analyzing Algorithm

- ***Running time*** for a particular input is the number of *primitive operations* executed
- ***Assumption***: Constant time c_i for the execution of the i^{th} line (of pseudocode)

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $\quad key = A[j]$	c_2	$n-1$
3 $\quad$ // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $\quad i = j - 1$	c_4	$n-1$
5 $\quad$ **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $\quad\quad A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $\quad\quad i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $\quad A[i+1] = key$	c_8	$n-1$

Note: t_j is the number of times the ***while*** loop test in line 5 is executed for the value of j.

Analyzing Algorithm

The running time of the algorithm is

$$\sum_{\text{all statements}} (\text{cost of statement}) \cdot (\text{number of times statement is executed})$$

Let $T(n)$ = running time of INSERTION-SORT.

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\ &\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) . \end{aligned}$$

Analyzing Algorithm

- ***Best case***
  - Array is already sorted, so $t_j = 1$ for $j = 2, 3, \ldots, n$.
  - $T(n) = c_1 n + c_2(n - 1) + c_4(n - 1) + c_5(n - 1) + c_8(n - 1)$
    $= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$
    $=$ <u>$an + b$ *(linear in n)*</u>

- ***Worst case***
  - $$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

Analyzing Algorithm

- **Average Case?**
- Concentrate on <u>worst-case</u> running time
  - Provides the upper bound
  - Occurs often
  - Average case is often as bad as the worst case
- ***Order of Growth***
  - The order of a running-time function is the fastest growing term, discarding constant factors
  - Insertion sort
    - Best case: $an + b \quad \rightarrow \Theta(n)$
    - Worst case: $an^2 + bn + c \quad \rightarrow \Theta(n^2)$

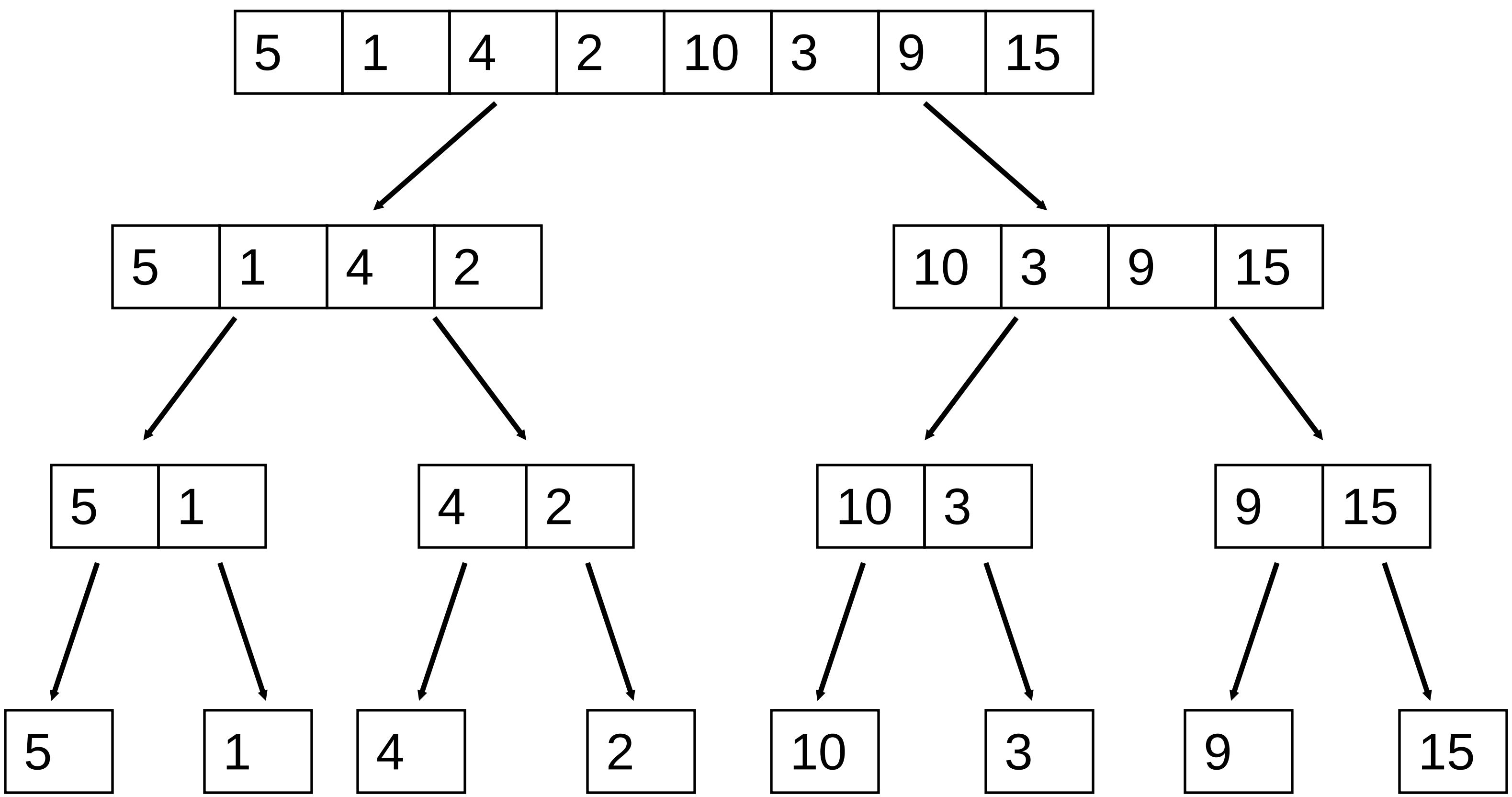
Designing Algorithms

- *Incremental* design
  - Iterative
  - Example: insertion sort
- *Divide-and-conquer* algorithm
  - Recursive
  - Example: merge sort
- Three steps in the divide-and-conquer paradigm
  - *Divide* the problem into smaller subproblems
  - *Conquer* subproblems by solving them recursively
  - *Combine* solutions of subproblems

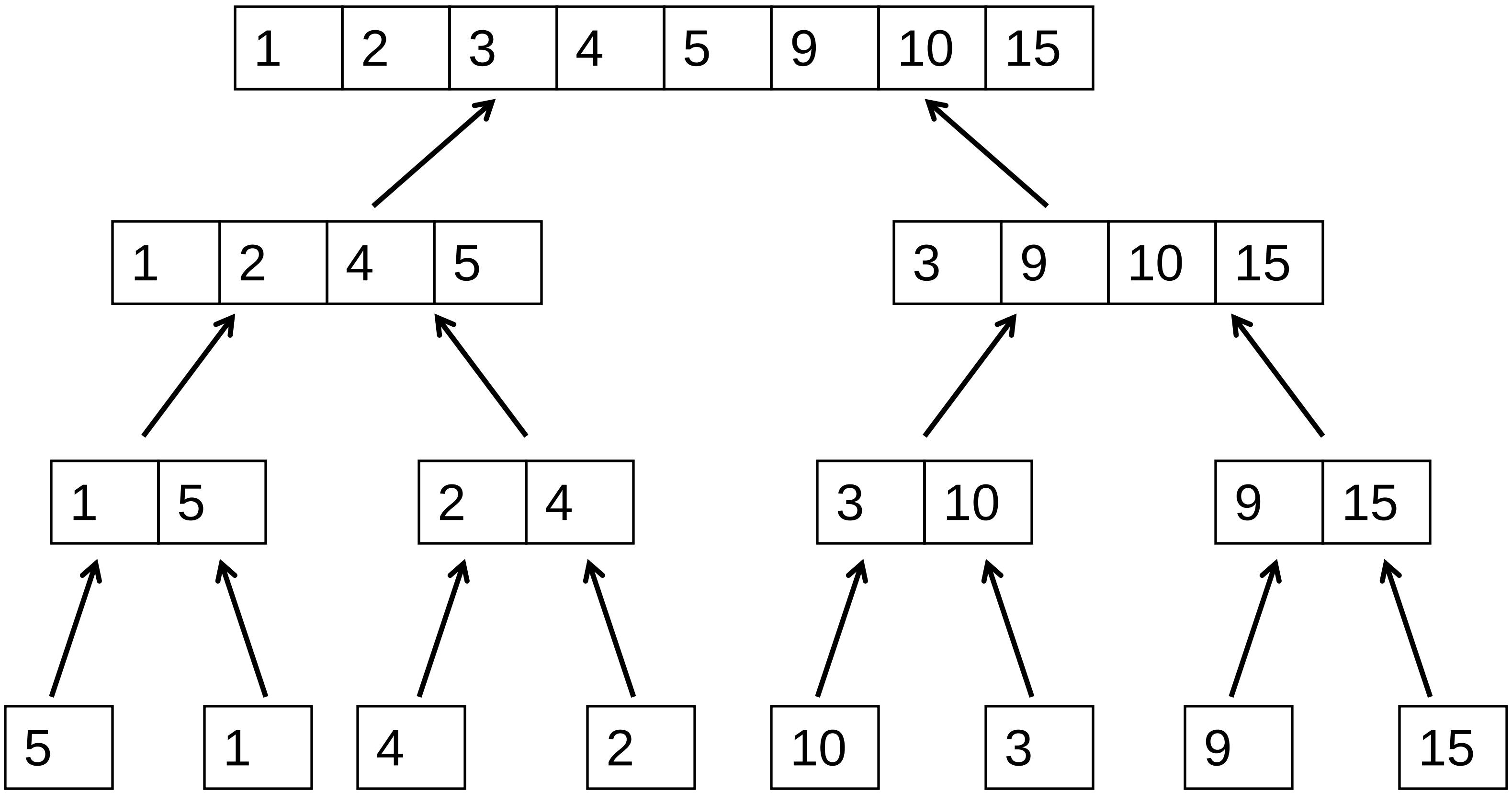
Designing Algorithms

- ***Merge Sort***
  - *Divide* the n-element sequence into two subsequences of $n/2$ elements each
  - *Conquer* (sort) the two subsequences recursively using merge sort
  - *Combine* (merge) the two sorted subsequences to produce the sorted answer
- Note: Recursion *bottoms out* when only one element to be sorted

Divide ...

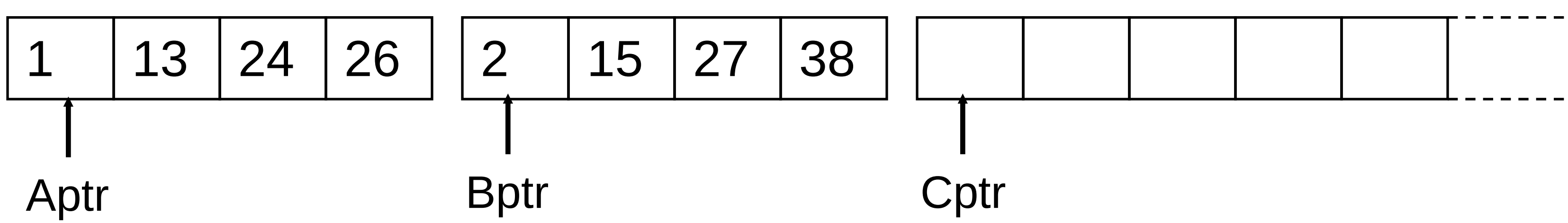


And Conquer

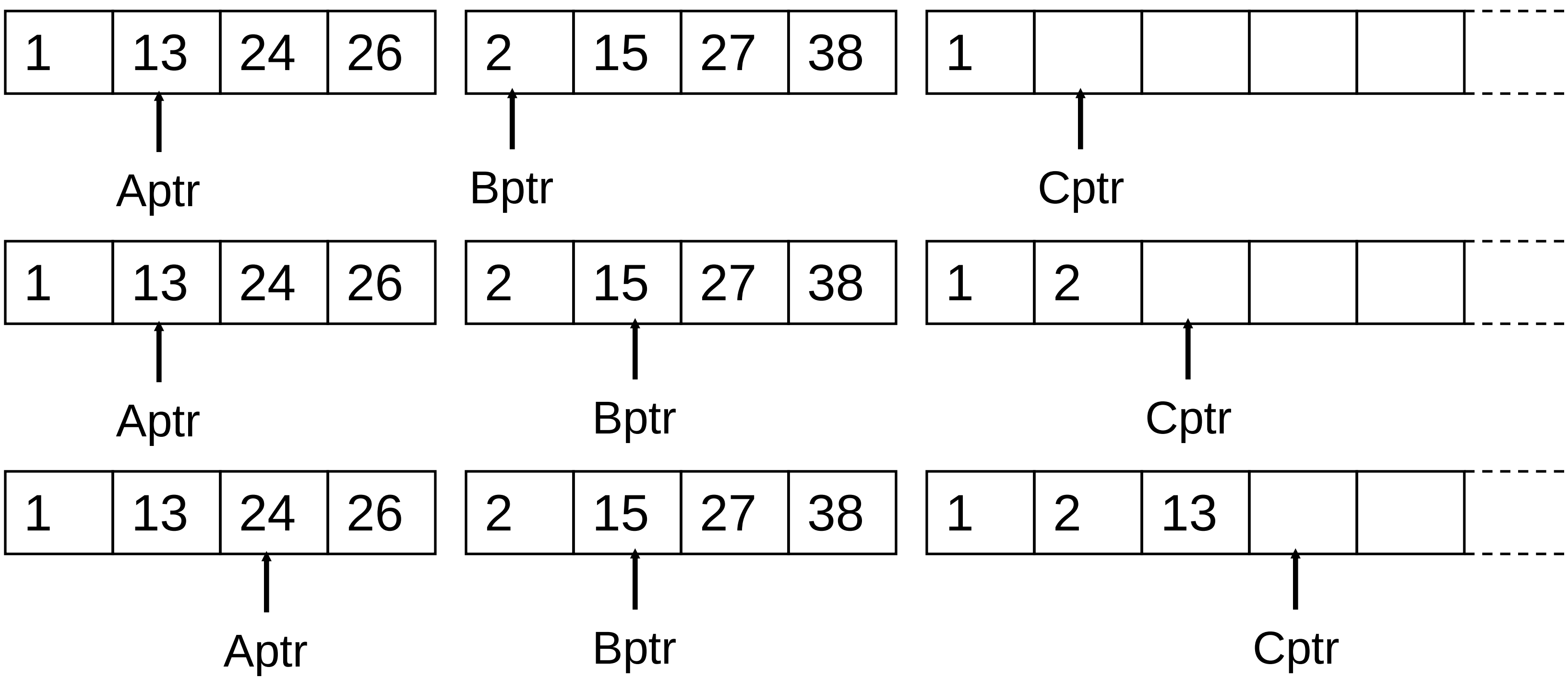


Merge Sort

- For MERGE_SORT, an initial array is repeatedly divided into halves (usually each is a separate array), until arrays of just one element remain
- At each level of recombination, two sorted arrays are merged into one
- This is done by copying the smaller of the two elements from the sorted arrays into the new array, and then moving along the arrays



Merging


1 13 24 26
2 15 27 38
1
Aptr
Bptr
Cptr
1 13 24 26
2 15 27 38
1 2
Aptr
Bptr
Cptr
1 13 24 26
2 15 27 38
1 2 13
Aptr
Bptr
Cptr

etc.

Merge Sort Algorithm

- MERGE_SORT(A, p, r)

if p < r

then q ← $\lfloor (p + r) / 2 \rfloor$

    MERGE_SORT(A, p, q)

    MERGE_SORT(A, q+1, r)

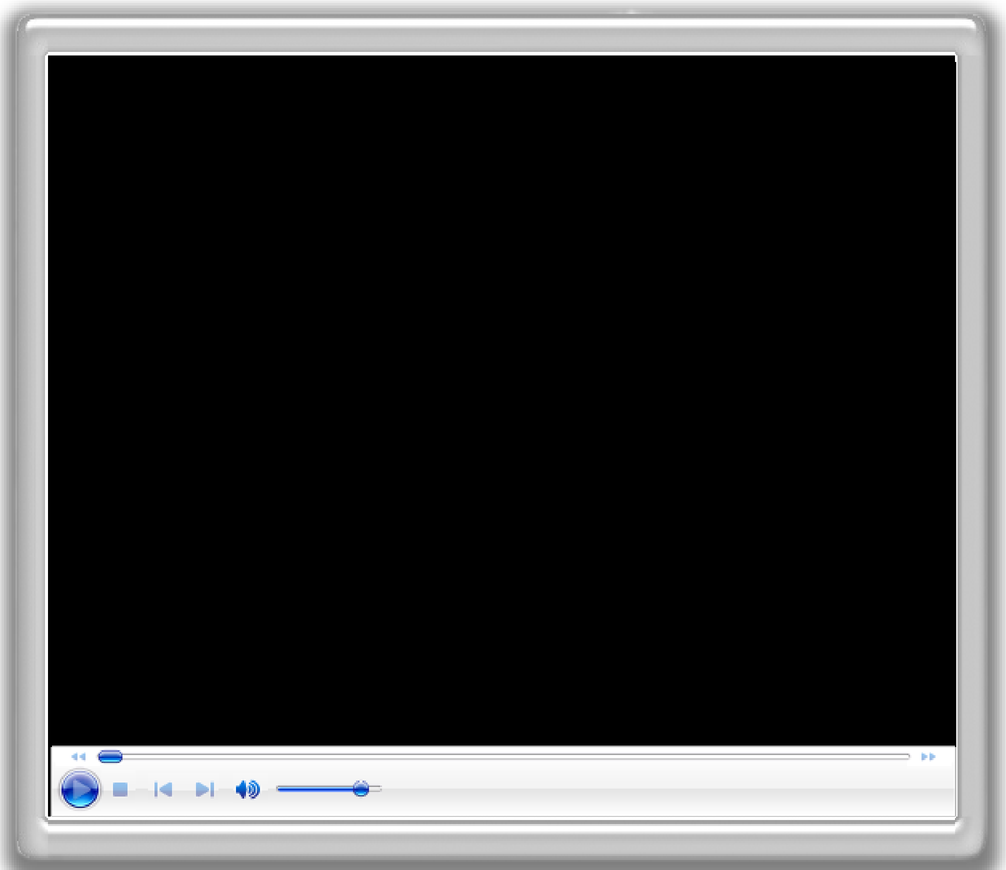
    MERGE(A, p, q, r)

```cpp
#include <cstdlib>
#include <iostream>
using namespace std;
int a[50];
void merge(int low,int mid,int high)
{
 int h,i,j,b[50],k;
 h=low; i=low; j=mid+1;

 while((h<=mid)&&(j<=high))
 {
  if(a[h]<=a[j]) { b[i]=a[h]; h++; }
  else { b[i]=a[j]; j++; }
  i++;
 }
 if(h>mid)
  for(k=j;k<=high;k++)
  {
    b[i]=a[k]; i++;
  }
 else
  for(k=h;k<=mid;k++)
  {
    b[i]=a[k]; i++;
  }
 }
 for(k=low;k<=high;k++)
  a[k]=b[k];
}
void merge_sort(int low,int high)
{
 int mid;
 if(low<high)
 {
  mid=(low+high)/2;
  merge_sort(low,mid);
  merge_sort(mid+1,high);
  merge(low,mid,high);
 }
}
int main()
{
 int num,i;

 cout<<"Enter the NUMBER OF ELEMENTS:"<<endl;
 cin>>num;
 cout<<endl;
 cout<<"Enter ELEMENT ( "<< num <<" ):"<<endl;
 for(i=1;i<=num;i++) cin>>a[i] ;

 merge_sort(1,num);

 cout<<endl;
 cout<<"The sorted list will be :"<<endl;
 for(i=1;i<=num;i++) cout<<a[i]<<"  ";

 return 0;
}
```

Merge Sort Algorithm

- Video Content
  - An illustration of Merge Sort with Transylvanian-saxon (German) folk dance.

Merge Sort Algorithm



Merge Sort Algorithm

- Note:
  - The MERGE_SORT(A, p, r) sorts the elements in the subarray A[p .. r]
  - If $p >= r$, the subarray has at most one element and is therefore already sorted
  - The procedure MERGE(A, p, q, r), where $p <= q < r$, merges two already sorted subarrays A[p ..q] and A[$q+1$ .. r]. It takes $\Theta(n)$ time
  - To sort an array A[1 .. n], we call MERGE_SORT(A, 1, n)

Analyzing Divide-And-Conquer Algorithms

- Running time is described by a ***recurrence equation*** or ***recurrence***
- Assume:
  - A problem is divided into a subproblems, each of which is $1/b$ the size of the original
  - Dividing the problem takes $D(n)$ time
  - Combining the solutions to subproblems into the solution to the original problem takes $C(n)$ time
- $T(n) = \Theta(1)$ if $n <= c$,
  $= aT(n/b) + D(n) + C(n)$ otherwise.

Analyzing Divide-And-Conquer Algorithms

- ***Analysis of Merge Sort***
  - ***Divide***: Computes the middle of the subarray $D(n) = \Theta(1)$
  - ***Conquer***: We recursively solve two subproblems, each of size $n/2$, contributing $2T(n/2)$
  - ***Combine***: The MERGE procedure takes $\Theta(n)$, so, $C(n) = \Theta(n)$
- The worst-case ***running time*** of merge sort is:
  - $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Thanks to contributors

Mr. Pham Van Nguyen (2022)

Dr. Thien-Binh Dang (2017 – 2022)

Prof. Hyunseung Choo (2001 – 2022)

Copyright 2000-2022 IT Faculty