Algorithms and Problem Solving

- Instructor: Zahra Derakhshandeh

What is an algorithm?

What is an algorithm?

An algorithm may be broadly defined as:

- a **step by step procedure** for solving a problem or accomplishing some end.

- It is a finite sequence of **unambiguous**, **executable** steps that ultimately **terminates** if followed.

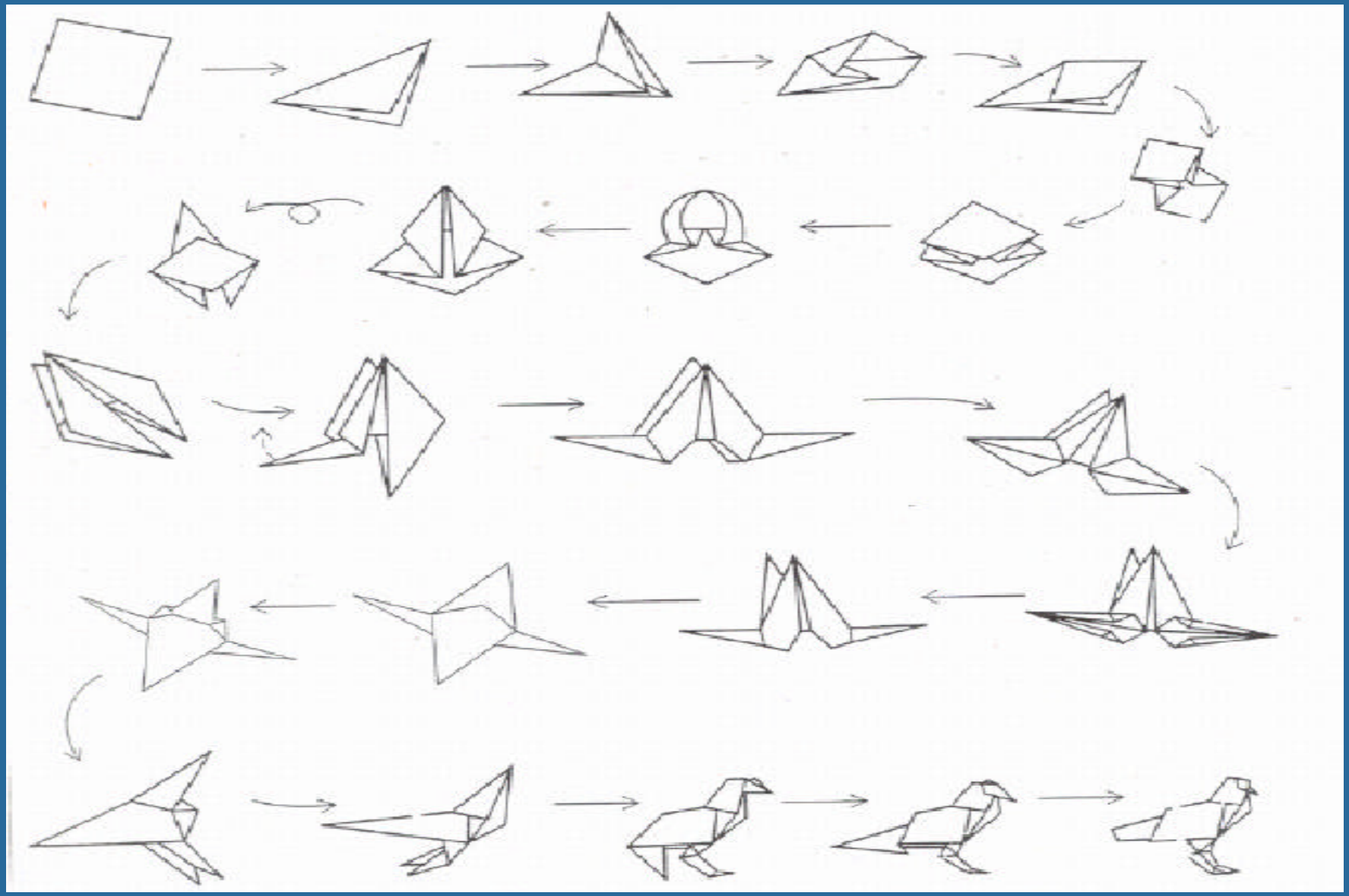
What is an algorithm?

An algorithm may be broadly defined as:

- a step by step procedure for solving a problem or accomplishing some end.
- It is a finite sequence of unambiguous, executable steps that ultimately terminate if followed.

- Is this an algorithm?
  1. Start
  2. Make a list of all positive integers
  3. Arrange this list in descending order (from largest to smallest)
  4. Extract the first integer from the resulting list
  5. Stop.

Example in Origami: Algorithm for making a bird



Algorithms = Problem Solving

Example : Find Max: Write an algorithm to find the largest (integer) number among three different numbers entered by user.

Algorithms = Problem Solving

Example : Find Max

1. Start
2. Consider three given numbers and declare them as a, b, and c.
1. If a>b
        If a>c
            Display a as the largest
        else Display c is the largest
2. else
        if b>c
        Display b as the largest
        else Display c is the largest
3. Stop.

Algorithms = Problem Solving

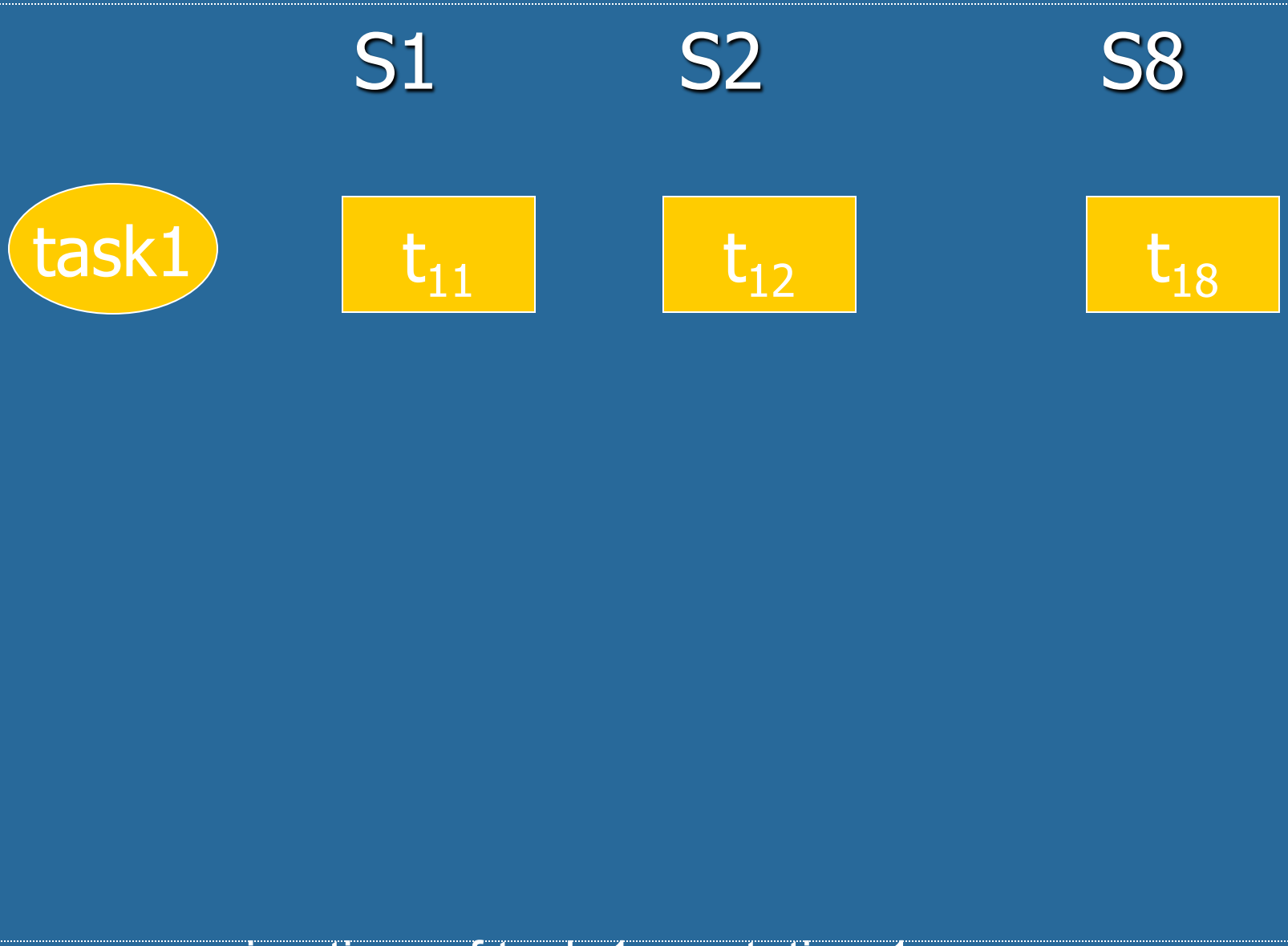
Example : Searching

You are given a list of numbers and you would like to search the list and see whether a specific number is in the list or not.

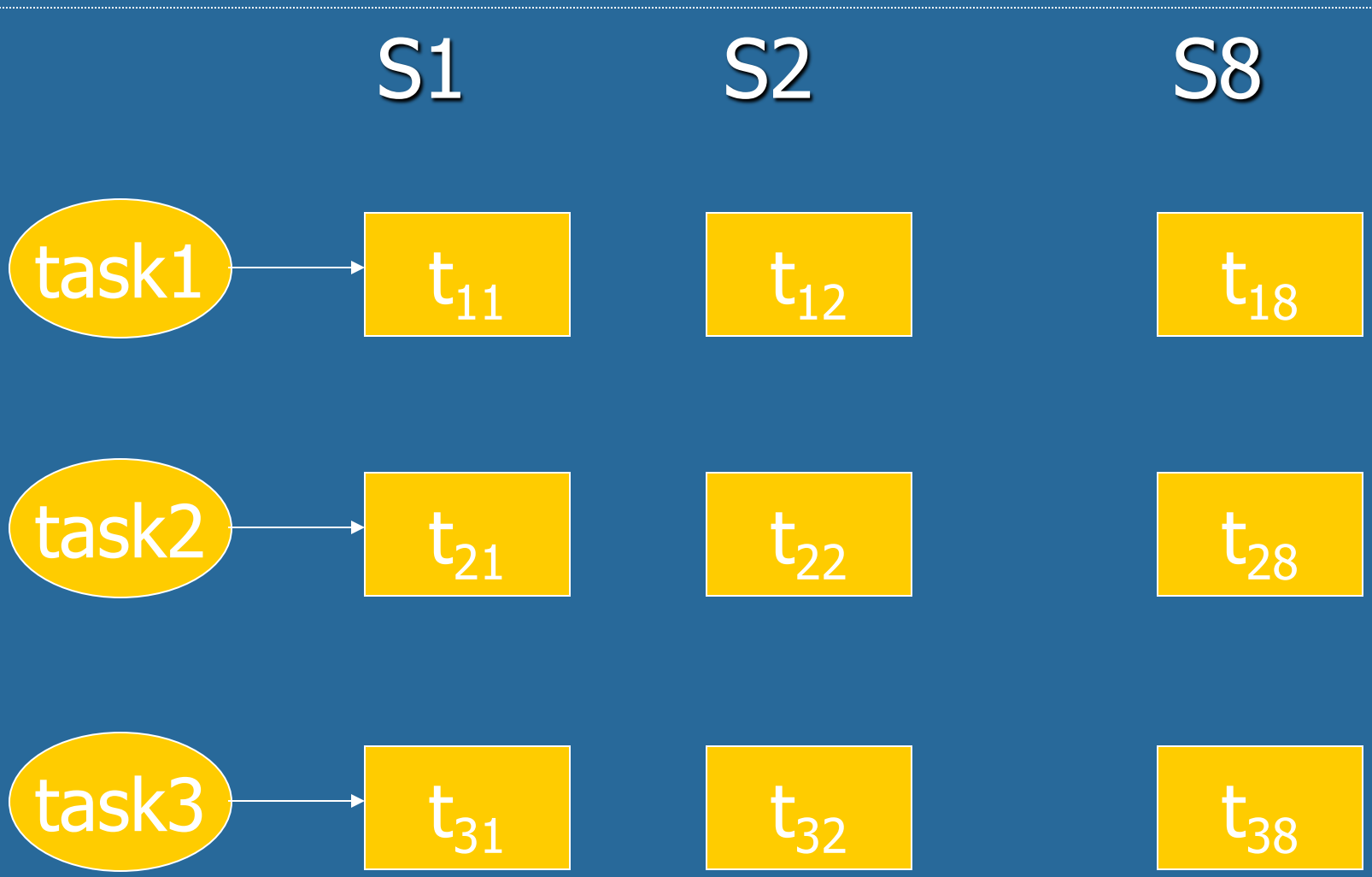
Algorithms = Problem Solving

Example in Manufacturing:

- Various tasks are to be processed in a series of stations. The processing time of the tasks in different stations is different.



t_{11}: processing time of task 1 on station 1



t_{11}: processing time of task 1 on station 1

Algorithms = Problem Solving

Example in Manufacturing:

- Various tasks are to be processed in a series of stations. The processing time of the tasks in different stations is different.
- Once a task is processed on a station it needs to be processed on the next station immediately, i.e., there cannot be any wait.

In what order should the tasks be supplied to the assembly line so that the completion time of processing of all tasks is minimized?

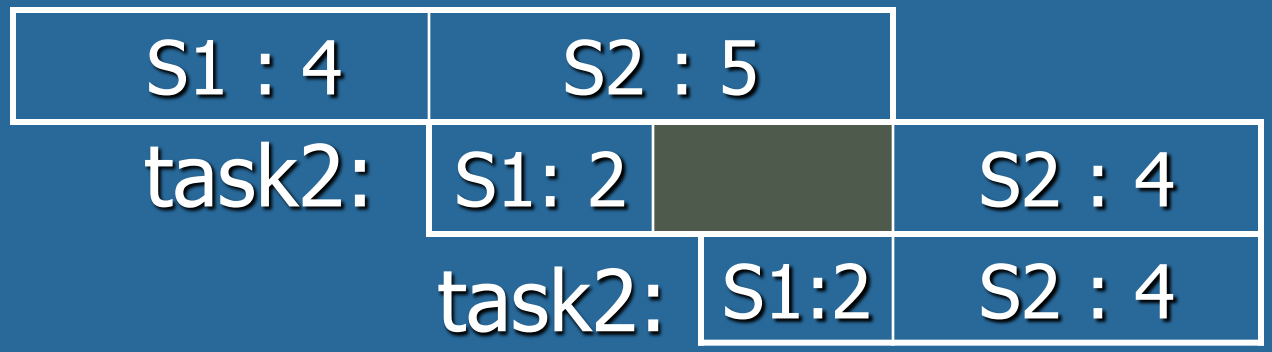
task1 : $t_{11} = 4,\ t_{12} = 5;$

task2 : $t_{21} = 2,\ t_{22} = 4;$

S1, S2

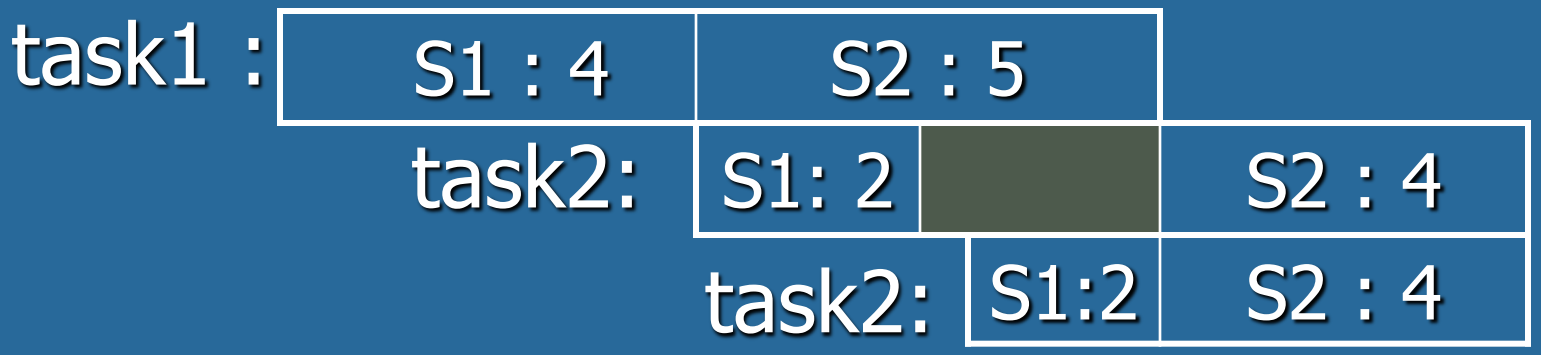
task1 : $t_{11} = 4,\ t_{12} = 5;$

task2 : $t_{21} = 2,\ t_{22} = 4;$



task1 : $t_{11} = 4,\ t_{12} = 5;$

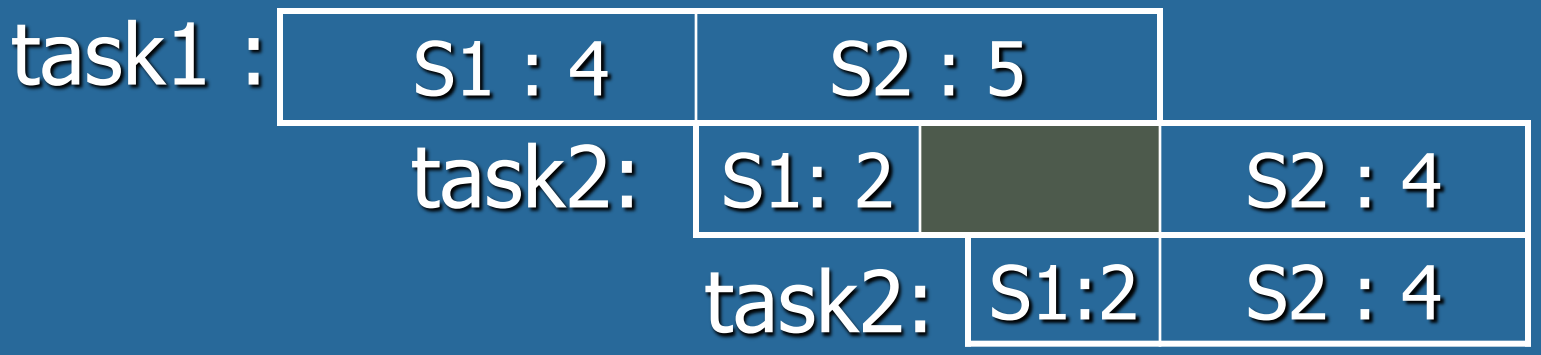
task2 : $t_{21} = 2,\ t_{22} = 4;$



Completion Time in this ordering = ?

task1 : $t_{11} = 4,\ t_{12} = 5;$

task2 : $t_{21} = 2,\ t_{22} = 4;$

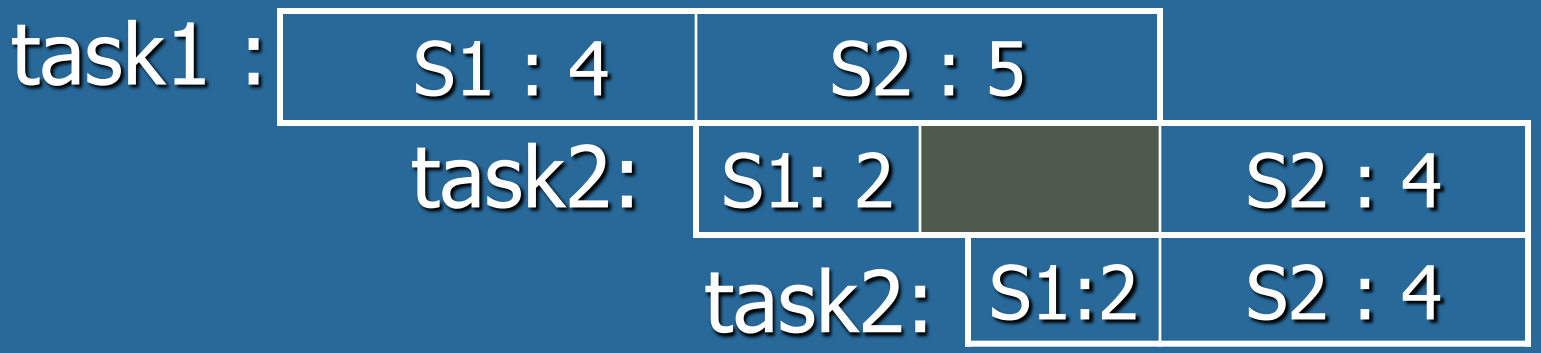


Completion Time in this ordering = 13

Can we make it better?

task1 : $t_{11} = 4,\ t_{12} = 5;$

task2 : $t_{21} = 2,\ t_{22} = 4;$

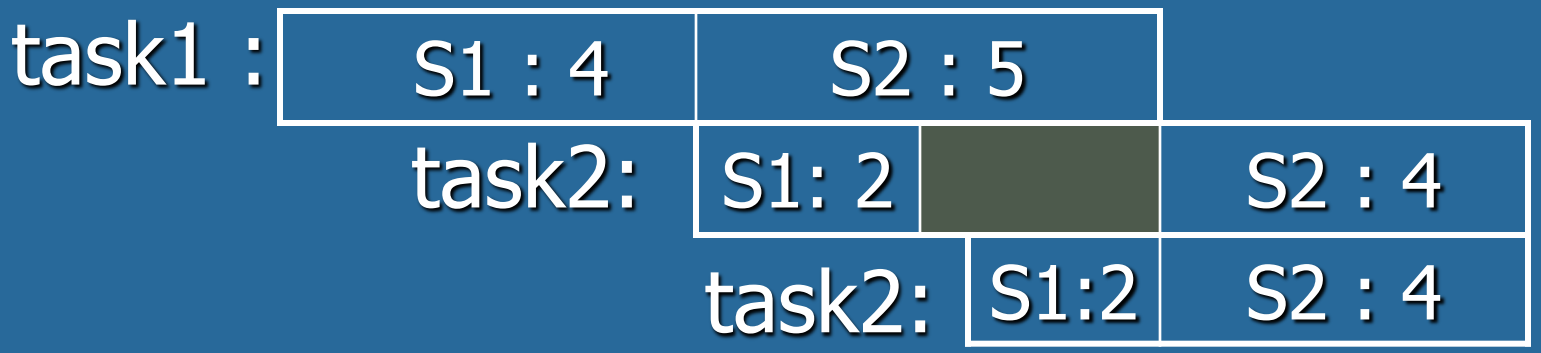


Completion Time in the first ordering = 13

Completion Time in the second ordering = ?

task1 : $t_{11} = 4,\ t_{12} = 5;$

task2 : $t_{21} = 2,\ t_{22} = 4;$



Completion Time in the first ordering = 13

Completion Time in the second ordering = 11

Problem and Instance

- Algorithms are designed to solve problems
  - What is a problem?

Problem and Instance

- Algorithms are designed to solve problems
  - What is a problem?
    - A problem is a general question to be answered, usually processing several parameters, or free variables, whose values are left unspecified.
    - A problem is described by giving

      (i) a general description of all its parameters and

      (ii) a statement of what properties the answer, or the solution, required to satisfy.

Problem and Instance

- Algorithms are designed to solve problems
  - A problem is described by giving

    (i) a general description of all its parameters and

    (ii) a statement of what properties the answer, or the solution, required to satisfy.

  - What is an instance?

Problem and Instance

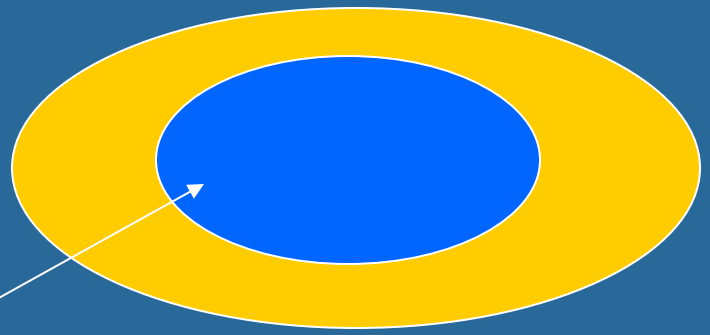
- Algorithms are designed to solve problems
  - A problem is described by giving

    (i) a general description of all its parameters and

    (ii) a statement of what properties the answer, or the solution, required to satisfy.

- What is an instance?
  - An instance of a problem is obtained by specifying particular values for all the problem parameters.

Search Space



- The solution is somewhere here
  - Solution can be found by exhaustive search in the search space
- Search space for the solution may be very large
- Does large search space imply long computation time to find solution?
- Not necessarily true
- Search space for the sorting problem is very large
- The trick in the design of **efficient algorithms** lies in finding **ways to reduce the search space**

Properties of Algorithms

- Finiteness: An algorithm must always terminate after a finite number of steps
- Definiteness: Each step must be precisely defined; the actions must be unambiguous
- Input: An algorithm has zero or more inputs
  - Offline Algorithms: ?
  - Online Algorithms: ?

Properties of Algorithms

- Finiteness: An algorithm must always terminate after a finite number of steps
- Definiteness: Each step must be precisely defined; the actions must be unambiguous
- Input: An algorithm has zero or more inputs
  - Offline Algorithms: All input data is available before the execution of the algorithm begins
  - Online Algorithms: Input data is made available during the execution of the algorithm

Properties of Algorithms

- Finiteness: An algorithm must always terminate after a finite number of steps
- Definiteness: Each step must be precisely defined; the actions must be unambiguous
- Input: An algorithm has zero or more inputs

- Output: An algorithm has one or more outputs
- Efficiency
  - Def. An algorithm is efficient if its running time is polynomial.

Evaluating Quality of Algorithms

- Often there are several different ways to solve a problem, i.e., there are several different algorithms to solve a problem
- What is the "best" way to solve a problem?
- What is the "best" algorithm?
- How do you measure the "goodness" of an algorithm?
- What metric(s) should be used to measure the "goodness" of an algorithm?
  - **Time**
  - Space

Measuring efficiency of algorithms

- One possible way to measure efficiency may be to note the **execution time** on some machine
- Suppose that the problem P can be solved by two different algorithms A1 and A2.
- Algorithms A1 and A2 were coded and using a data set D, the programs were executed on some machine M
- A1 and A2 took 10 and 15 seconds to run to completion
- Can we now say that A1 is more efficient that A2?

Measuring efficiency of algorithms

- What happens if instead of data set D we use a different dataset D'?
  - A1 may end up taking more time than A2
- What happens if instead of machine M we use a different machine M'?
  - A1 may end up taking more time than A2
- If one wants to make a statement about the efficiency of two algorithms based on timing values, it should read "**A1 is more efficient that A2 on machine M, using data set D**", instead of an unqualified statement like "A1 is more efficient that A2"

Measuring efficiency of algorithms

- The qualified statement "A1 is more efficient that A2 on machine M, using data set D" is of limited value as someone may use different data set or a different machine
- **Ideally**, one would like to make an unqualified statement like "**A1 is more efficient that A2**" , that is independent of data set and machine
- We **cannot** make such an unqualified statement by observing execution time on a machine
- Data and Machine independent statement can be made if we note the number of "**basic operations**" needed by the algorithms
  - The "basic" or "elementary" operations are operations of the form addition, multiplication, comparison etc.

Big-O Notation

- The best way is to compare algorithms by the amount of work done in a critical loop, as a function of the number of input elements (*N*)
- **Big-O:** A notation expressing execution time (complexity) as the term in a function that increases most rapidly relative to *N*
- Consider the *order of magnitude* of the algorithm

Common Orders of Magnitude

- O(1): Constant or *bounded* time; not affected by *N* at all
- O($\log_2 N$): Logarithmic time; each step of the algorithm cuts the amount of work left in half
- O(*N*): Linear time; each element of the input is processed
- O($N \log_2 N$): $N \log_2 N$ time; apply a logarithmic algorithm N times or vice versa

Common Orders of Magnitude (cont.)

- $O(N^2)$: Quadratic time; typically apply a linear algorithm *N* times, or process every element with every other element
- $O(N^3)$: Cubic time; naive multiplication of two NxN matrices, or process every element in a three-dimensional matrix
- $O(2^N)$: Exponential time; computation increases dramatically with input size

What About Other Factors?

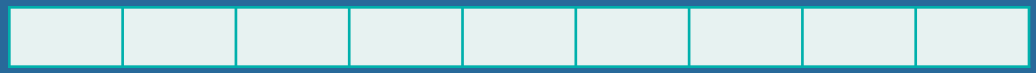
- Consider f(N) = $2N^4 + 100N^2 + 10N + 50$
- We can ignore $100N^2 + 10N + 50$ because $2N^4$ grows so quickly
- Similarly, the 2 in $2N^4$ does not greatly influence the growth
- The final order of magnitude is O(N^4)
- The other factors may be useful when comparing two very similar algorithms

Example: Searching algorithms

- Linear Search
- Binary Search

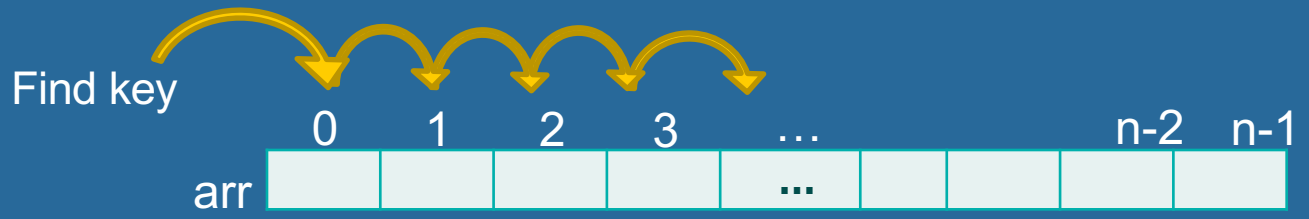
Example: Searching

Find an item in a sorted array of n items.



List of items/values

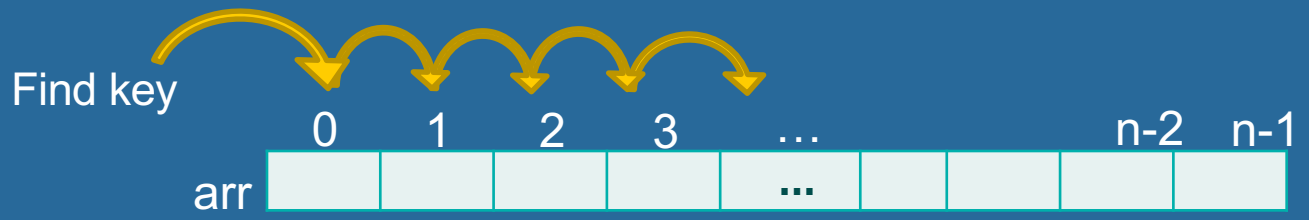
Linear Search



List of n integer values

```
int search(int arr[], int n, int key)
{
    int i;
    for(i=0; i<n; i++)
        if(arr[i] == key)
         return i;
    return -1;
}
```

Linear Search

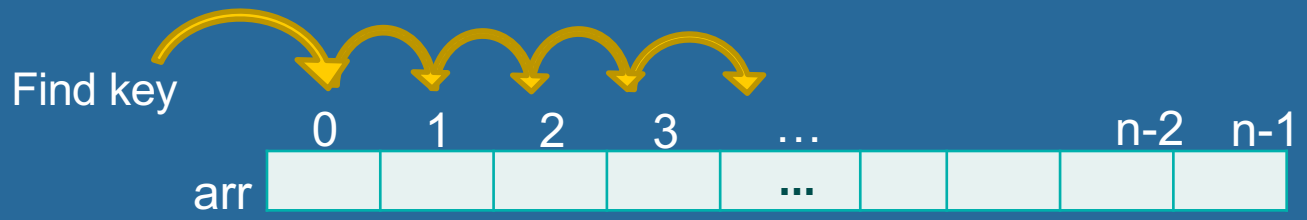


List of n integer values

```
int search(int arr[], int n, int key)
{
    int i;
    for(i=0; i<n; i++)
        if(arr[i] == key)
         return i;
    return -1;
}
```

# of basic operations?

#of Basic operations in Linear Search



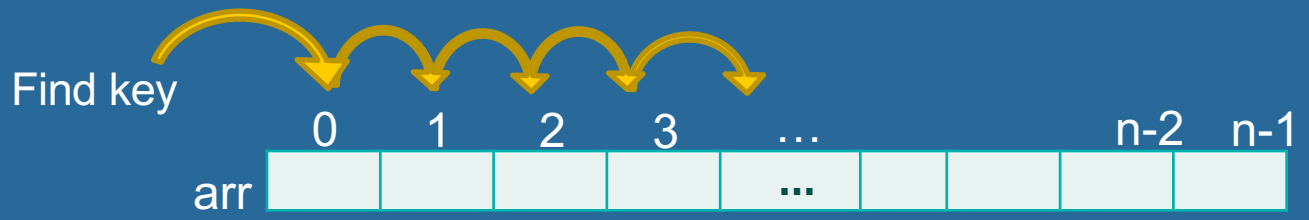
List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0 ; i<n; i++)

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1

#of Basic operations in Linear Search



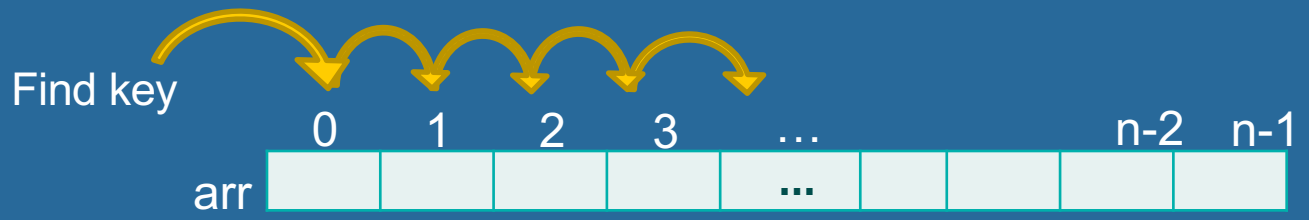
List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0 ; i<n;  i++)

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1
c_2	n+1

#of Basic operations in Linear Search



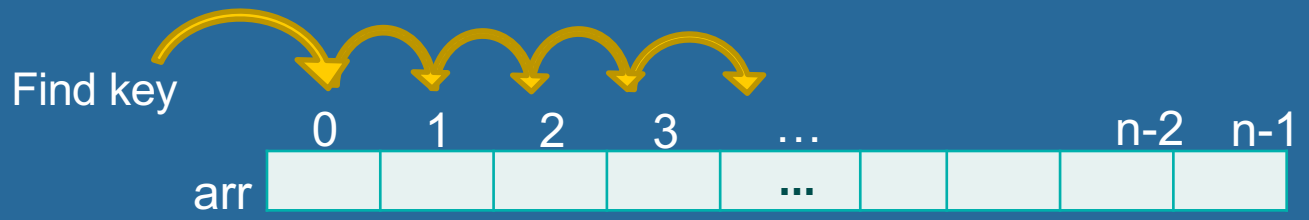
List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0; i<n ; i++ )

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1
c_2	n+1
c_3	n

#of Basic operations in Linear Search



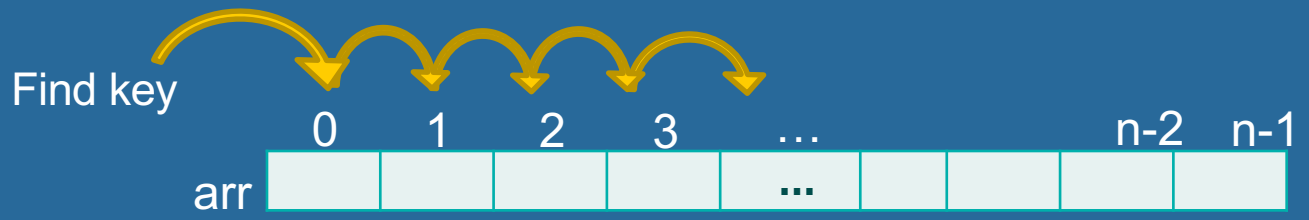
List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0; i<n ; i++)

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1
c_2	n+1
c_3	n
c_4	?

#of Basic operations in Linear Search



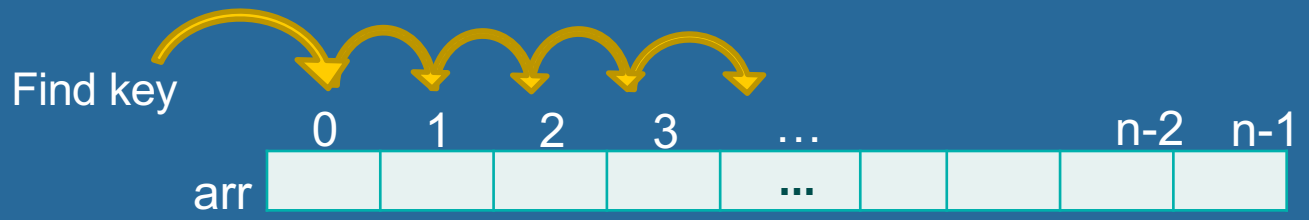
List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0; i<n ; i++)

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1
c_2	n+1
c_3	n
c_4	n

#of Basic operations in Linear Search



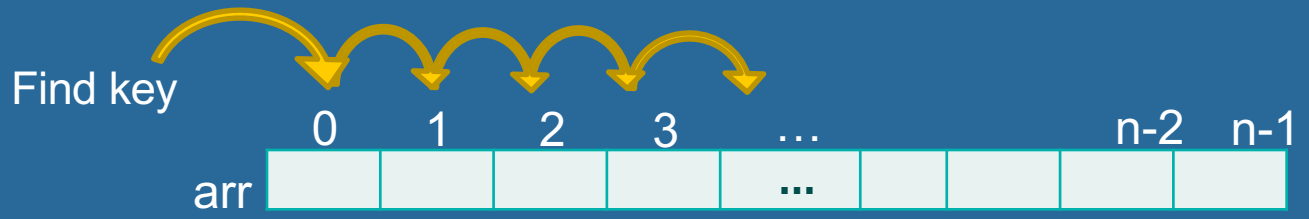
List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0; i<n ; i++)

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1
c_2	n+1
c_3	n
c_4	n
c_5	1

#of Basic operations in Linear Search



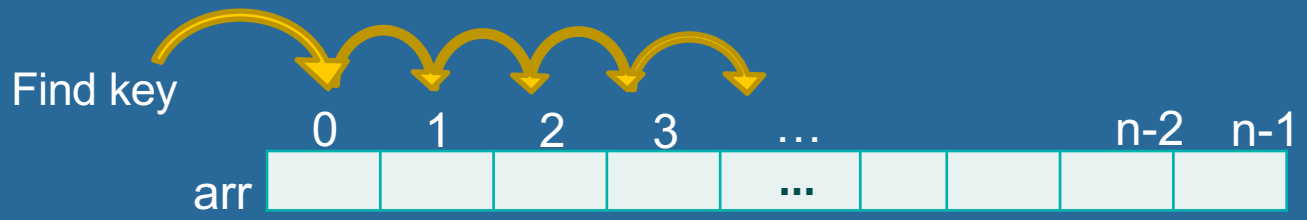
List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0; i<n ; i++)

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1
c_2	n+1
c_3	n
c_4	n
c_5	1
c_6	1

#of Basic operations in Linear Search



List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0; i<n ; i++)

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1
c_2	n+1
c_3	n
c_4	n
c_5	1
c_6	1

$f(n)= (c_2+c_3+c_4)n + c_1+c_2+c_5+c_6$

#of Basic operations in Linear Search

- Goal: Given a name, find the matching phone number in the phone book
- Algorithm 1: Linear search through the phone book until the name is found

- $f(n)= (c_2+c_3+c_4)n + c_1+c_2+c_5+c_6 = C.n+C'$

- The smaller factor is essentially ignorable as n grows

- Worst case: $f(n)= O(n)$

Analysis

Best case complexity:
- When we are very lucky
- The smallest number of steps that an algorithm can take
- Not very useful analysis

Worst case complexity:
- The greatest number of steps that an algorithm may require

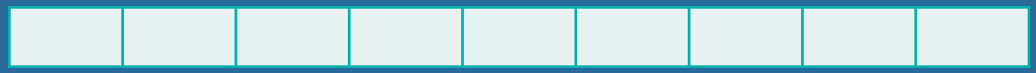
Average case complexity
- Average number of steps required, considering all possible inputs
- The most difficult one to compute

Example: Phone Book Search

- Goal: Given a name, find the matching phone number in the phone book
- Algorithm 1: Linear search through the phone book until the name is found
- Best case: O(1) (it's the first name in the book)
- Worst case: O(*n*) (it's the final name)
- Average case: The name is near the middle, requiring *n*/2 steps, which is O(*n*)

Binary Search

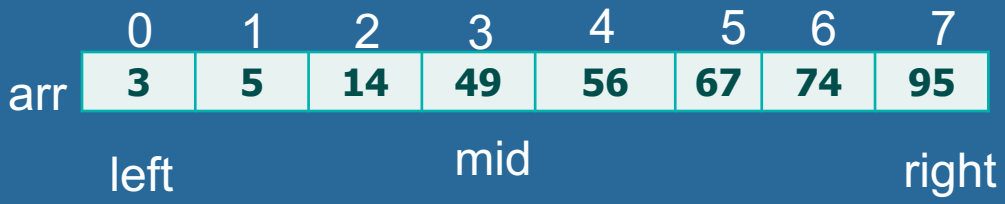
Find an item in a sorted array of n items.



List of items/values

Binary Search

Find key in the sorted list



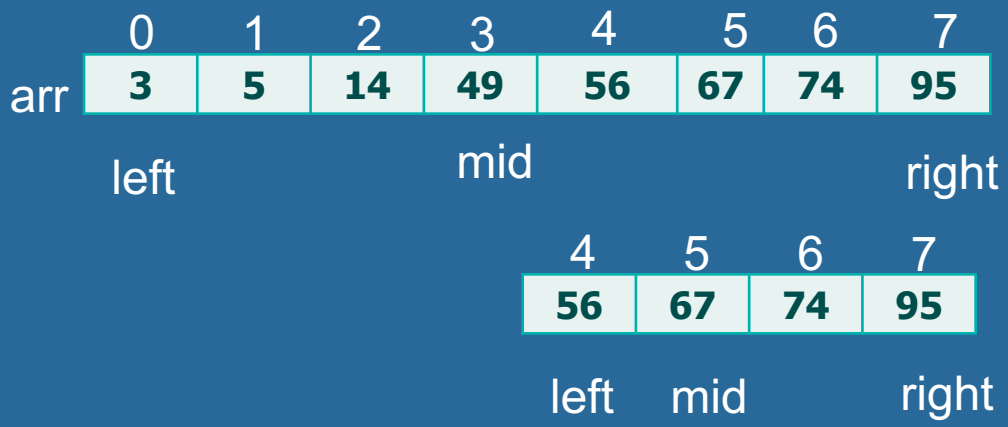

```
int binarySearch(int arr[], int left, int right, int key)
{ if(right>= left)
   {
        int mid = (left + right)/2;
        if(arr[mid] == key)
              return mid;
        if(arr[mid]>key)
              return binarySearch(arr, left, mid-1, key);
        return binarySearch(arr, mid+1, right, key);
   }
     return -1;}
```

Binary Search

Find key=56

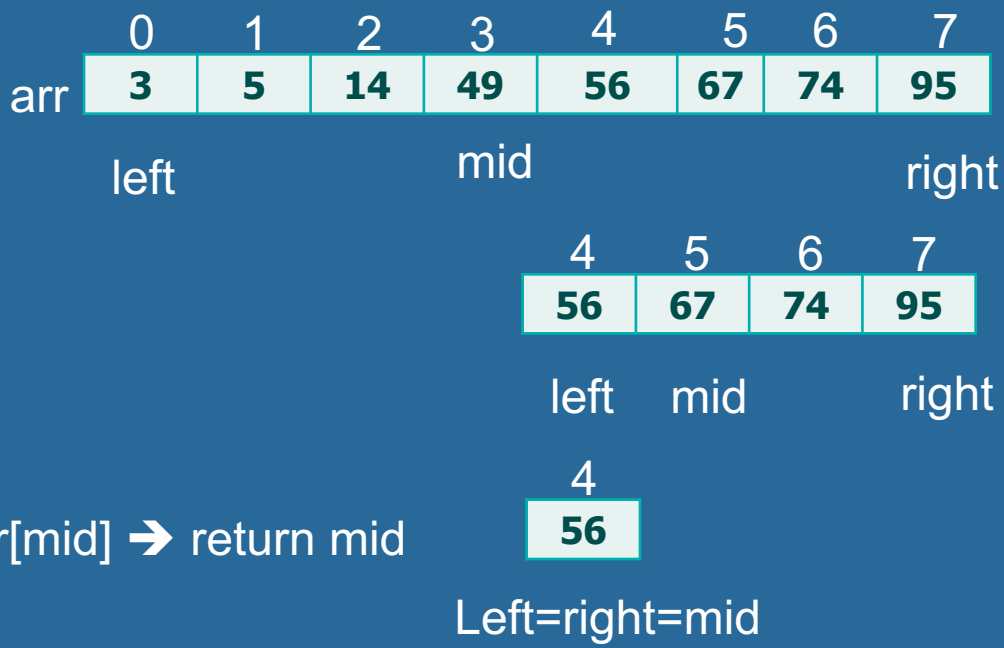
Binary Search

Find key=56



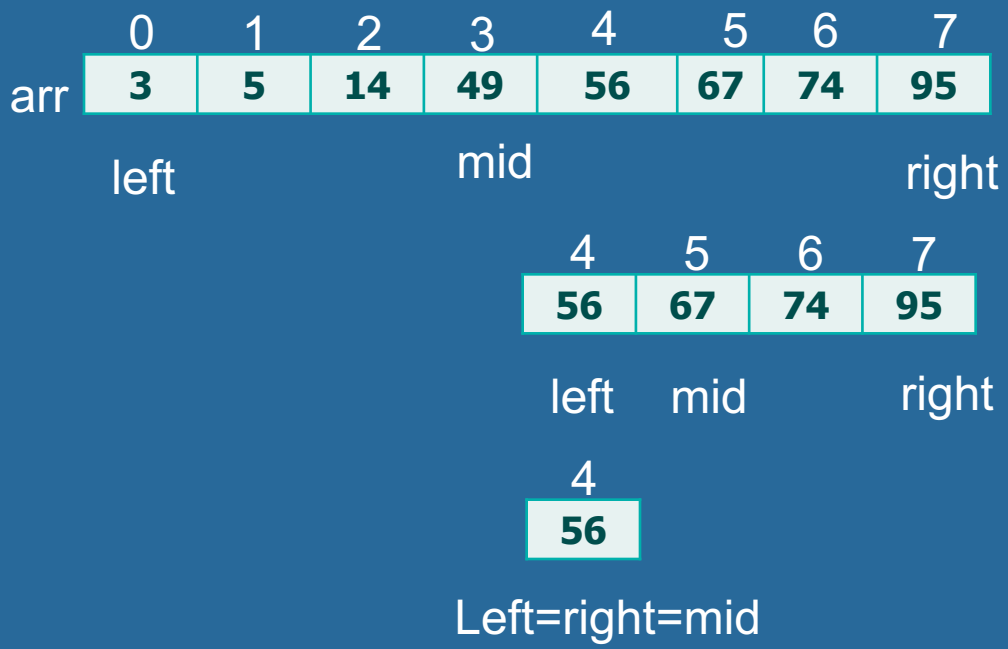
Binary Search

Find key=56



Binary Search

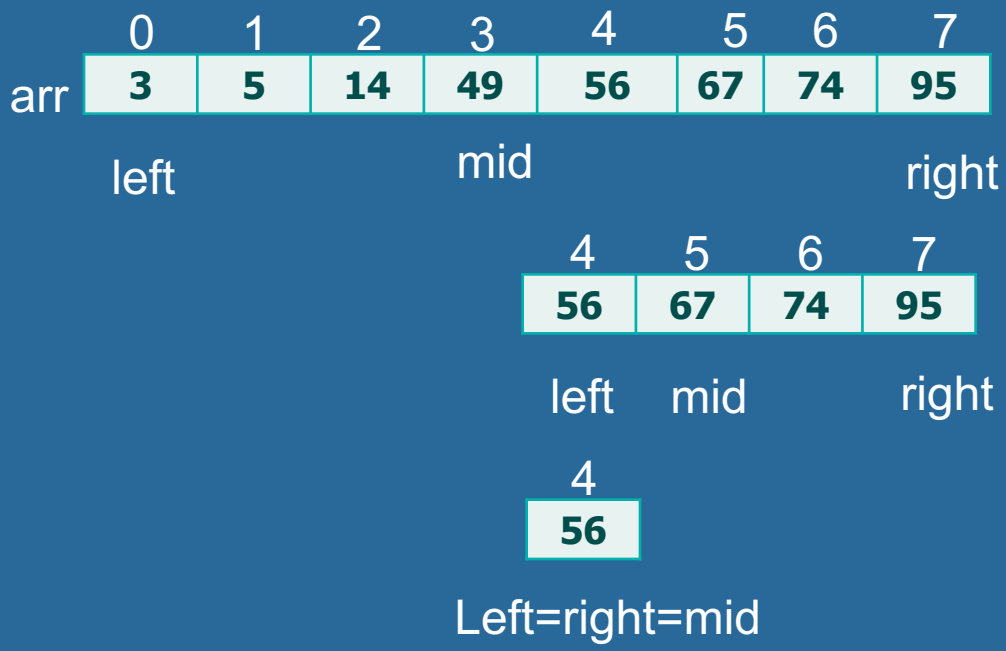
Find key=56



Binary Search time complexity: ?

Binary Search

Find key=56



log(n) for n=8

f(n)=c*log(n)

Binary Search time complexity: O(log n)

Example: Phone Book Search (Summary)

Algorithm 2: Since the phone book is sorted, we can use a more efficient search

1) Check the name in the middle of the book
2) If the target name is less than the middle name, search the first half of the book
3) If the target name is greater, search the last half
4) Continue until the name is found

Example: Phone Book Search (cont.)

Algorithm 2 Characteristics:

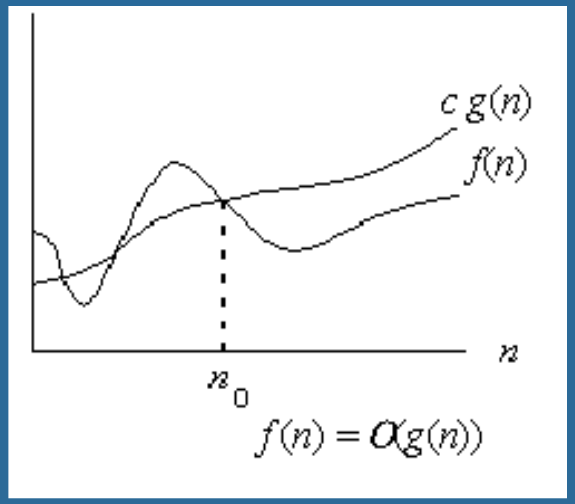
- Each step reduces the search space by half
- Best case: O(1) (we find the name immediately)
- Worst case: $O(\log_2 N)$ (we find the name after cutting the space in half several times)
- Average case: $O(\log_2 N)$ (it takes a few steps to find the name)

Elephants and Goldfish

- Think about buying elephants and goldfish and comparing different pet suppliers
- The price of the goldfish is trivial compared to the cost of the elephants
- Similarly, the growth from $100N^2 + 10N + 50$ is trivial compared to $2N^4$
- The smaller factors are essentially noise

Growth of Functions: Asymptotic Notations

Def.: We say $f(n)=O(g(n))$, "f is of the order of function g", if for some constants c and n_0, $f(n) \leq cg(n)$ for all $n \geq n_0$.



Upper Bound

$O(g(n))$ = {f(n): there exists positive constants c and n_0 such that $f(n) \leq c * g(n)$ for all $n \geq n_0$}

Example: upper bound?

- $f(n) = 2n^2 + 4n - 1$

- $f(n)=O(?)$
- Or $g(n)=?$

Example

- $f(n) = 2n^2 + 4n - 1$

$f(n)=O(n^2)$

$g(n)= n^2$, $c=10$, $n_0=?$

Example

- $f(n) = 2n^2 + 4n - 1$

$f(n)=O(n^2)$,

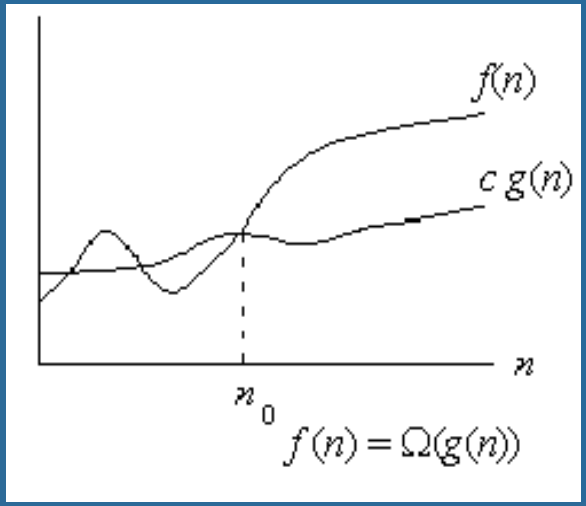
$g(n)= n^2$, $c=10$

$c=7$, $n_0=1$

$2n^2 + 4n - 1 <= 2n^2 + 4n^2 + n^2 <= 7n^2$

Growth of Functions: Asymptotic Notations

- $\Omega(g(n)) = \{f(n)$: there exists positive constants c and n_0 such that $cg(n) \leq f(n)$ for all $n \geq n_0\}$



Lower Bounds

Example: lower bound?

- $f(n) = 2n^2 + 4n - 1$
- $g(n)=?$

Example

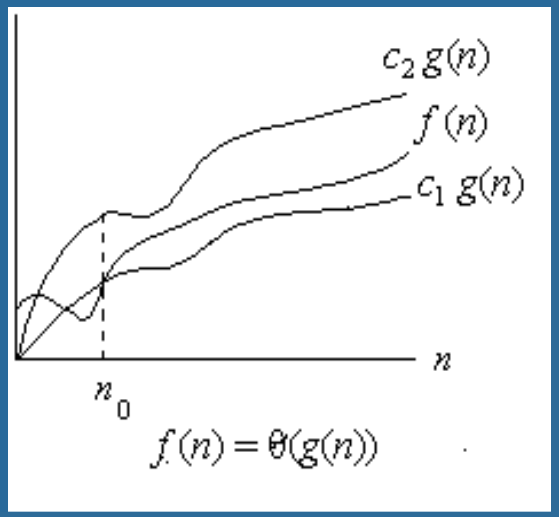
- $f(n) = 2n^2 + 4n - 1$
- $g(n) = n^2$

$$f(n) = \Omega(n^2)\ ,\ c=1,\ n_0=1$$

- $2n^2 + 4n - 1 >= n^2$

Growth of Functions: Asymptotic Notations

- $\Theta(g(n)) = \{f(n)$: there exists positive constants c_1, c_2 and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$



Tight Bounds

Example

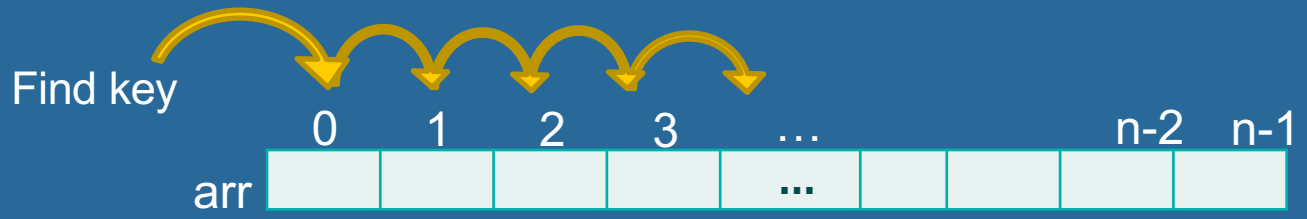
- $f(n) = 2n^2 + 4n - 1$

$f(n) = \Theta(n^2)$

Example

- Ex: $T(n) = 32n^2 + 17n + 32$.
  - $T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.
  - $T(n)$ is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$.

#of Basic operations in Linear Search



List of n integer values

```
int search(int arr[], int n, int key)
{
    for( int i=0; i<n ; i++)

        if(arr[i] == key)
            return i;
    return -1;
}
```

cost	times
c_1	1
c_2	n+1
c_3	n
c_4	n
c_5	1
c_6	1

$f(n)= (c_2+c_3+c_4)n + c_1+c_2+c_5+c_6$

Time complexity for Linear Search

$f(n)= (c_2+c_3+c_4)n + c_1+c_2+c_5+c_6$

Let constant $c> 2(c_1+c_2+c_3+c_4+c_5+c_6)$

Therefore, $f(n) \leq cn$ for $n \geq 1$

Therefore according to definition, $f(n)=O(n)$ for all $n \geq 1$ and constant c

Linear Search worst-case time complexity: O(n)