

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda Expressions

Searching. Sorting. Lambdas

Lect. PhD. Arthur Molnar

Babes-Bolyai University

arthur@cs.ubbcluj.ro

Overview

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- 1 Searching
 - The searching problem
 - Searching algorithms
 - Binary search
 - Search in Python
- 2 Sorting
 - The sorting problem
 - Selection sort
 - Insertion sort
 - Bubble Sort
 - Quick Sort
- 3 Lambda Expressions

Feedback for the course

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- You can write feedback at **academicinfo.ubbcluj.ro**
- It is both important as well as anonymous
- Write both what you like (so we keep&improve it) and what you don't
- Best if you write about all activities (lecture, seminar and laboratory)

Searching

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

- Data are available in the internal memory, as a sequence of records (k_1, k_2, \dots, k_n)
- Search a record having a certain value for one of its fields, called the **search key**.
- If the search is successful, we have the position of the record in the given sequence.
- We approach the search problem's two possibilities separately:
 - Searching with unordered keys
 - Searching with ordered keys

Searching - unordered keys

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

Problem specification

- **Data:** $a, n, (k_i, i = 0, \dots, n - 1)$, where $n \in \mathbb{N}, n \geq 0$.
- **Results:** p , where $(0 \leq p \leq n - 1, a = k_p)$ or $p = -1$, if key is not found.

Searching - unordered keys

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search
Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda
Expressions

```
def searchSeq(el, l):  
    '''  
        Search for an element in list  
        el – element  
        l – list of elements  
        Return the position of the element, -1 if not  
            found  
    '''  
    poz = -1  
    for i in range(0, len(l)):  
        if el == l[i]:  
            poz = i  
    return poz
```

Computational complexity is $T(n) = \sum_{i=0}^{n-1} 1 = n \in \Theta(n)$

Searching - unordered keys

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search
Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda
Expressions

```
def searchSeq(el, l):  
    '''  
    Search for an element in list  
    el - element  
    l - list of elements  
    Return the position of the element, -1 if not  
    found  
    '''  
    i = 0  
    while i < len(l) and el != l[i]:  
        i += 1  
    if i < len(l):  
        return i  
    return -1
```

What is the difference between this and the previous version?

Searching - unordered keys

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search
Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda
Expressions

- Best case: the element is at the first position,
 $T(n) \in \Theta(1)$.
- Worst case: the element is in the $n-1$ position,
 $T(n) \in \Theta(n)$.
- Average case: if distributing the element uniformly, the loop can be executed $0, 1, \dots, n-1$ times, so
 $T(n) = \frac{1+2+\dots+n-1}{n} \in \Theta(n)$.
- Overall complexity is $O(n)$

Searching - ordered keys

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

Problem specification

- **Data:** $a, n, (k_i, i = 0, \dots, n - 1)$, where $n \in \mathbb{N}, n \geq 0$, and $k_0 < k_1 < \dots < k_{n-1}$;
- **Results:** p , where $(p = 0 \text{ and } a \leq k_0)$ or $(p = n \text{ and } a > k_{n-1})$ or $(0 < p \leq n - 1 \text{ and } (k_{p-1} < a \leq k_p))$.

Searching - ordered keys

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search
Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda
Expressions

```
def searchSeq(el, l):  
    '''  
    Search for an element in list  
    el - element  
    l - list of ordered elements  
    Return the position of the first occurrence, or  
    position where element can be inserted  
    '''  
  
    if len(l) == 0: return 0  
    poz = -1  
    for i in range(0, len(l)):  
        if el <= l[i]:  
            poz = i  
    if poz == -1: return len(l)  
    return poz
```

Computational complexity is $T(n) = \sum_{i=0}^{n-1} 1 = n \in \Theta(n)$

Searching - ordered keys

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search
Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

```
def searchSuccesor(el, l):  
    '''  
    Search for an element in list  
    el – element  
    l – list of ordered elements  
    Return the position of the first occurrence, or  
    position where element can be inserted  
    '''  
    if len(l)==0 or el<=l[0]:  
        return 0  
    if el>=l[-1]:  
        return len(l)  
    i = 0  
    while i<len(l) and el>l[i]:  
        i += 1  
    return i
```

Searching - ordered keys

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search
Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda
Expressions

- Best case: the element is at the first position,
 $T(n) \in \Theta(1)$.
- Worst case: the element is in the $n-1$ position,
 $T(n) \in \Theta(n)$.
- Average case: if distributing the element uniformly, the loop can be executed $0, 1, \dots, n-1$ times, so
 $T(n) = \frac{1+2+\dots+n-1}{n} \in \Theta(n)$.
- Overall complexity is $O(n)$

Searching algorithms

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

- *Sequential search*
 - Keys are successively examined
 - Keys may not be ordered
- *Binary search*
 - Uses the divide and conquer technique
 - Keys are ordered

Recursive binary-search algorithm

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

```
def binarySearch(key, data, left, right):  
    '''  
    Search for an element in an ordered list  
    key – element to search  
    left, right – bounds of the search  
    Return insertion position of key that keeps list  
        ordered  
    '''  
    if left >= right - 1:  
        return right  
    middle = (left + right) // 2  
    if key < data[middle]:  
        return binarySearch(key, data, left, middle)  
    else:  
        return binarySearch(key, data, middle, right)  
    )  
print(binarySearch(2000, data, 0, len(data)))
```

Recursive binary-search function

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

```
def search(key, data):  
    '''  
        Search for an element in an ordered list  
        key – element to search  
        data – the list  
        Return insertion position of key that keeps list  
            ordered  
    '''  
    if len(data) == 0 or key < data[0]:  
        return 0  
    if key > data[-1]:  
        return len(data)  
    return binarySearch(key, data, 0, len(data))
```

Binary-search recurrence

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

■ The recurrence:
$$T(n) = \begin{cases} 1, & n = 1 \\ T(\frac{n}{2}) + 1, & n > 1 \end{cases}$$

Iterative binary-search function

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda
Expressions

```
def binarySearch(key, data):  
    '''  
    - specification -  
    '''  
    if len(data) == 0 or key < data[0]:  
        return 0  
    if key > data[-1]:  
        return len(data)  
    left = 0  
    right = len(data)  
    while right - left > 1:  
        middle = (left + right) // 2  
        if key <= data[middle]:  
            right = middle  
        else:  
            left = middle  
    return right
```

Search problem runtime complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

Algorithm	Best case	Average	Worst case	Overall
Sequential	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Successor	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$O(n)$
Binary-search	$\Theta(1)$	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$	$O(\log_2 n)$

Demo

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem

Searching
algorithms

Binary search

Search in Python

Sorting

The sorting
problem

Selection sort

Insertion sort

Bubble Sort

Quick Sort

Lambda

Expressions

Collections and search

Examine the source code in **ex34_search.py**

The sorting problem

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

Sorting

Rearrange a data collection in such a way that the elements of the collection verify a given order.

- **Internal sort** - data to be sorted are available in the internal memory
- **External sort** - data is available as a file (on external media)
- **In-place sort** - transforms the input data into the output, only using a small additional space. Its opposite is called out-of-place.
- **Sorting stability** - we say that sorting is stable when the original order of multiple records having the same key is preserved

The sorting problem

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- Elements of the data collection are called records
- A record is formed by one or more components, called fields
- A key K is associated to each record, and is usually one of the fields.
- We say that a collection of n records is:
 - Sorted in increasing order by the key K : if $K(i) \leq K(j)$ for $0 \leq i < j < n$
 - Sorted in decreasing order: if $K(i) \geq K(j)$ for $0 \leq i < j < n$

Internal sorting

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

Problem specification

- **Data:** n, K , where $K = (k_1, k_2, \dots, k_n)$, $k_i \in \mathbb{R}, i = 1, n$
- **Results:** K' , where K' is a permutation of K , having sorted elements: $k'_1 \leq k'_2 \leq \dots \leq k'_n$.

Sorting algorithms

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda Expressions

A few algorithms that we will study:

- Selection sort
- Insertion sort
- Bubble sort
- Quick sort

Selection Sort

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- Determine the element having the minimal key, and swap it with the first element.
- Resume the procedure for the remaining elements, until all elements have been considered.

Selection sort algorithm

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

```
def selectionSort(data):  
    for i in range(-1, len(data)-1):  
        index = i  
        # Find smallest element in the rest of the  
        list  
        for j in range(i+1, len(data)):  
            if data[j] < data[index]:  
                index = j  
            if i < index:  
                data[i], data[index] = data[index],  
                data[i]
```

Selection sort - time complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- The total number of comparisons is

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1 = \frac{n(n-1)}{2} \in \Theta(n^2)$$

- Independent of the input data size, what are the best, average, worst-case computational complexities?

Selection sort - space complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- **In-place** algorithms. Algorithms that use a small (constant) quantity of additional memory.
- **Out-of-place** or not-in-space algorithms. Algorithms that use a non-constant quantity of extra-space.
- The additional memory required by selection sort is $O(1)$.
- Selection sort is an in-place sorting algorithm.

Direct selection sort

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

```
def directSelectionSort(data):  
    for i in range(0, len(data) - 1):  
        # Select the smallest element  
        for j in range(i + 1, len(data)):  
            if data[j] < data[i]:  
                data[i], data[j] = data[j], data[i]
```

Direct selection sort

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

■ Overall time complexity: $\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1 = \frac{n(n-1)}{2} \in \Theta(n^2)$

Insertion Sort

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- Traverse the elements.
- Insert the current element at the right position in the subsequence of already sorted elements.
- The sub-sequence containing the already processed elements is kept sorted, so that, at the end of the traversal, the whole sequence is sorted.

Insertion Sort - Algorithm

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

```
def insertSort(data):  
    for i in range(1, len(data)):  
        index = i - 1  
        elem = data[i]  
        # Insert into correct position  
        while index >= 0 and elem < data[index]:  
            data[index + 1] = data[index]  
            index -= 1  
        data[index + 1] = elem
```

Insertion Sort - time complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- Maximum number of iterations (worst case) happens if the initial array is sorted in a descending order:

$$T(n) = \sum_{i=2}^n (i-1) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

Insertion Sort - time complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- Minimum number of iterations (best case) happens if the initial array is already sorted:

$$T(n) = \sum_{i=2}^n 1 = n - 1 \in \Theta(n)$$

Insertion Sort - Space complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- Time complexity - The overall time complexity of insertion sort is $O(n^2)$.
- Space complexity - The complexity of insertion sort is $\theta(1)$
- Insertion sort is an **in-place** sorting algorithm.

Bubble Sort

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- Compares pairs of consecutive elements that are swapped if not in the expected order.
- The comparison process ends when all pairs of consecutive elements are in the expected order.

Bubble Sort - Algorithm

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

```
def bubbleSort(data):  
    done = False  
    while not done:  
        done = True  
        for i in range(0, len(data) - 1):  
            if data[i] > data[i+1]:  
                data[i], data[i+1] = data[i+1], data  
                    [i]  
                done = False
```

Bubble Sort - Complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- **Best-case** running time complexity order is $\theta(n)$
- **Worst-case** running time complexity order is $\theta(n^2)$
- **Average** running-time complexity order is $\theta(n^2)$
- **Space complexity**, additional memory required is $\theta(1)$
- Bubble sort is an *in-place* sorting algorithm.

Quick Sort

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

Based on the *divide and conquer* technique

- 1 **Divide:** partition array into 2 sub-arrays such that elements in the lower part \leq elements in the higher part.

Partitioning

Re-arrange the elements so that the element called pivot occupies the final position in the sub-sequence. If i is that position: $k_j \leq k_i \leq k_l$, for $Left \leq j < i < l \leq Right$

- 2 **Conquer:** recursively sort the 2 sub-arrays.
- 3 **Combine:** trivial since sorting is done in place.

Quick Sort - partitioning algorithm

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

```
def partition(data, left, right):  
    pivot = data[left]  
    i = left  
    j = right  
    while i != j:  
        # Find an element smaller than the pivot  
        while data[j] >= pivot and i < j:  
            j -= 1  
        data[i] = data[j]  
        # Find an element larger than the pivot  
        while data[i] <= pivot and i < j:  
            i += 1  
        data[j] = data[i]  
    # Place the pivot in position  
    data[i] = pivot  
    return i
```

Quick Sort - algorithm

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

```
def quickSort(data, left, right):  
    # Partition the list  
    pos = partition(data, left, right)  
    # Order left side  
    if left < pos - 1:  
        quickSort(data, left, pos - 1)  
    # Order right side  
    if pos + 1 < right:  
        quickSort(data, pos + 1, right)
```


Quick Sort - time complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- The run time of quick-sort depends on the distribution of splits
- The partitioning function requires linear time
- **Best case**, the partitioning function splits the array evenly: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$, $T(n) \in \Theta(n \log_2 n)$

Quick Sort - best partitioning

Lecture 12

Lect. PhD.
Arthur Molnar

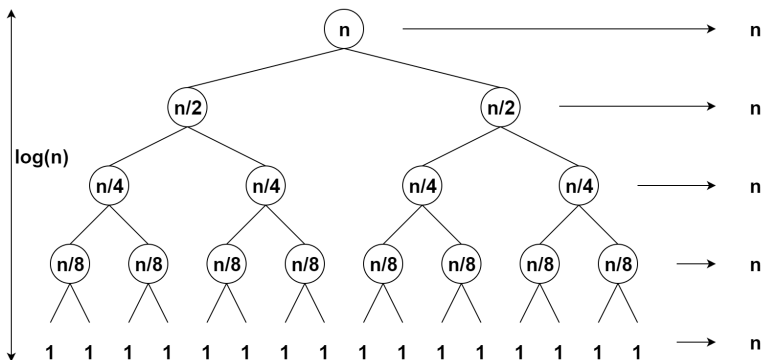
Searching

The searching problem
Searching algorithms
Binary search
Search in Python

Sorting

The sorting problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions



- We partition n elements $\log_2 n$ times, so
 $T(n) \in \Theta(n \log_2 n)$

Quick Sort - worst partitioning

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

- In the worst case, function Partition splits the array such that one side of the partition has only one element:

$$T(n) = T(1) + T(n-1) + \Theta(n) = T(n-1) + \Theta(n) = \sum_{k=1}^n \Theta(k) \in \Theta(n^2)$$

Quick Sort - Worst case

Lecture 12

Lect. PhD.
Arthur Molnar

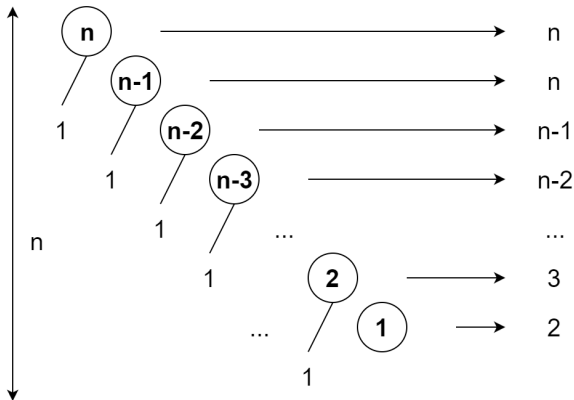
Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions



- Worst case partitioning appears when the input array is sorted or reverse sorted, so n elements are partitioned n times, $T(n) \in \Theta(n^2)$

Sorting runtime complexity

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

Algorithm	Worst case	Average
Selection sort	$\Theta(n^2)$	$\Theta(n^2)$
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Bubble sort	$\Theta(n^2)$	$\Theta(n^2)$
Quick sort	$\Theta(n^2)$	$\Theta(n \log_2 n)$

Demo

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda
Expressions

Sorting

Examine the source code in **ex35_sort.py**

Lambda expressions

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

The searching
problem
Searching
algorithms
Binary search
Search in Python

Sorting

The sorting
problem
Selection sort
Insertion sort
Bubble Sort
Quick Sort

Lambda Expressions

Lambda expressions

Small anonymous functions, that you define and use in the same place.

- Syntactically restricted to a single expression.
- Can reference variables from the containing scope (just like nested functions).
- They are *syntactic sugar* for a function definition.

Demo

Lecture 12

Lect. PhD.
Arthur Molnar

Searching

- The searching problem
- Searching algorithms
- Binary search
- Search in Python

Sorting

- The sorting problem
- Selection sort
- Insertion sort
- Bubble Sort
- Quick Sort

Lambda Expressions

Lambda Expressions

Examine the source code in **ex36_lambdas.py**