**Outline**

**Topics**

* Review
* Algorithm techniques:
  + Brute Force
  + Decrease-and-Conquer
* Analyses tool(s): Big �

**Part 0: Review**

* Algorithm techniques:
  + Brute Force: [Linear search](https://www.cs.usfca.edu/~galles/visualization/Search.html)
  + Decrease-and-Conquer: [Binary Search](https://www.cs.usfca.edu/~galles/visualization/Search.html)
  + Divide-and-Conquer
* [Big $O$](https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/)
* [Recursion](https://recursion.vercel.app/) vs Iteration

**Grading/ Week (Points)**

* Week 1 (Max 10 points)
* Week 2 (Max 15 points)
* Week 3
  + Group Activities (Max 9 points)
  + Homework (Individual activities) (Max 9 points)
  + [Peer reviews (Max 7 points)](https://github.com/TT00FE39-3001/lecture3/blob/main/peer-assessment-template.xlsx)

**Part 1: Sorting using brute force**

* Algorithm techniques: **Brute Force**
  + Bubble sort
  + Selection sort
  + Big � analysis
* [Activity 1](https://github.com/TT00FE39-3001/lecture3/blob/main/activity1/README.md)

**Part 2: Sorting using decrease and conquer**

* Algorithm techniques: **decrease and conquer**
  + Insertion sort
  + Big � analysis
  + Average case vs worst case
* [Activity 2](https://github.com/TT00FE39-3001/lecture3/blob/main/activity2/README.md)

**Part 3: Introduction to Linked list**

* [Big O Complexity](https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1176/handouts/midterm/5-BigO.pdf)
* Introduction to linked list
  + Queues
  + Stack
* [Activity 3](https://github.com/TT00FE39-3001/lecture3/blob/main/activity3/README.md)

**Misc**

* [Links](https://github.com/TT00FE39-3001/lecture3/blob/main/links.md)
* [Cheat Sheet: Mathematical Notation in Markdown](https://upyesp.org/posts/makrdown-vscode-math-notation/)

**About**

*No description, website, or topics provided.*

**Resources**

[Readme](https://github.com/TT00FE39-3001/lecture3#readme)

**Stars**

[**0** stars](https://github.com/TT00FE39-3001/lecture3/stargazers)

**Watchers**

[**0** watching](https://github.com/TT00FE39-3001/lecture3/watchers)

**Forks**

[**0** forks](https://github.com/TT00FE39-3001/lecture3/network/members)

[**Releases**](https://github.com/TT00FE39-3001/lecture3/releases)

No releases published

[**Packages**](https://github.com/orgs/TT00FE39-3001/packages?repo_name=lecture3)

No packages published

**Languages**

* [C++89.1%](https://github.com/TT00FE39-3001/lecture3/search?l=c%2B%2B)
* [Python10.9%](https://github.com/TT00FE39-3001/lecture3/search?l=python)

**Footer**

© 2023 GitHub, Inc.

**Footer navigation**

* [Terms](https://docs.github.com/site-policy/github-terms/github-terms-of-service)
* [Privacy](https://docs.github.com/site-policy/privacy-policies/github-privacy-statement)
* [Security](https://github.com/security)
* [Status](https://www.githubstatus.com/)
* [D](https://docs.github.com/)

v**# Outline**

**## Topics**

- Review

- Algorithm techniques:

  - Brute Force

  - Decrease-and-Conquer

- Analyses tool(s): Big $O$

---

**## Part 0: Review**

- Algorithm techniques:

  - Brute Force: [Linear search](<https://www.cs.usfca.edu/~galles/visualization/Search.html>)

  - Decrease-and-Conquer: [Binary Search](<https://www.cs.usfca.edu/~galles/visualization/Search.html>)

  - Divide-and-Conquer

- [Big $O$](<https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/>)

- [Recursion](<https://recursion.vercel.app/>)

vs Iteration

**## Grading/ Week (Points)**

- Week 1 (Max 10 points)

- Week 2 (Max 15 points)

- Week 3

  - Group Activities (Max 9 points)

  - Homework (Individual activities) (Max 9 points)

  - [Peer reviews (Max 7 points)](./peer-assessment-template.xlsx)

---

**## Part 1: Sorting using brute force**

- Algorithm techniques: **\*\*Brute Force\*\***

  - Bubble sort

  - Selection sort

  - Big $O$ analysis

- [Activity 1](./activity1/README.md)

<!-- average case vs worst case -->

---

**## Part 2: Sorting using decrease and conquer**

- Algorithm techniques: **\*\*decrease and conquer\*\***

  - Insertion sort

  - Big $O$ analysis

  - Average case vs worst case

- [Activity 2](./activity2/README.md)

---

**## Part 3: Introduction to Linked list**

- [Big O Complexity](<https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1176/handouts/midterm/5-BigO.pdf>)

- Introduction to linked list

  - Queues

  - Stack

- [Activity 3](./activity3/README.md)

---

**## Misc**

- [Links](./links.md)

- [Cheat Sheet: Mathematical Notation in Markdown](<https://upyesp.org/posts/makrdown-vscode-math-notation/>)

links:

**# Links**

- [Data Structures In C++](<https://www.softwaretestinghelp.com/cpp-tutorials/>)

- [Visualization of Algorithms](<https://www.cs.usfca.edu/~galles/visualization>)

- [Data Structures and Algorithms](<https://www.techiedelight.com/>)

- [Understanding the formal definition of Big-O](<https://justin.abrah.ms/computer-science/understanding-big-o-formal-definition.html>)

- Bubble Sort

  - [Bubble-sort in C++](<https://www.softwaretestinghelp.com/bubble-sort/>)

  - [Bubble-sort Visualizer](<https://opendsa-server.cs.vt.edu/embed/bubblesortAV>)

  - [Bubble-sort tutorial](<https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/BubbleSort.html>)

- Selection Sort

  - [Selection-sort Visualizer](<https://opendsa-server.cs.vt.edu/ODSA/AV/Sorting/selectionsortAV.html>)

  - [Selection-sort In C++](<https://www.softwaretestinghelp.com/selection-sort/>)

  - [Selection-sort tutorial](<https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/SelectionSort.html>)

- [OpenDSA Data Structures and Algorithms Modules Collection Table Of Contents](<https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/index.html>)

- [Math support in Markdown](<https://github.blog/2022-05-19-math-support-in-markdown/>)

- [Cheat Sheet: Mathematical Notation in Markdown](<https://upyesp.org/posts/makrdown-vscode-math-notation/>)

- [Visualization: Queue based on linked list](<https://www.cs.usfca.edu/~galles/visualization/QueueLL.html>)

- [Visualization: Stack based on linked list](<https://www.cs.usfca.edu/~galles/visualization/StackLL.html>)

- <https://cpp.sh/>

HOMEWORK

**# Homework**

**## Task 1/3:Videos**

- [Sorts 1 Introduction to sorts (~8min)](<https://www.youtube.com/watch?v=H3FCoYQMKvI>)

- [Sorts 2 Selection Sort (~5min)](<https://youtu.be/fgYlVyrt1vE>)

- [Sorts 3 Insertion Sort (~5min)](<https://youtu.be/eTvQIbB-AuE>)

- [Sorts 4 Insertion Sort Code (~4min)](<https://youtu.be/3U2vTqaL7uE>)

-

**## Task 2/3: Reading**

- [Bubble Sort In C++ With Examples](https://www.softwaretestinghelp.com/bubble-sort/)

- [Selection Sort In C++ With Examples](https://www.softwaretestinghelp.com/selection-sort/)

- [Insertion Sort In C++ With Examples](https://www.softwaretestinghelp.com/insertion-sort/)

**## Task 3/3: Pre-Lecture**

- [Quicksort](https://youtu.be/0SkOjNaO1XY)

- [Pointer to pointer in c++](https://youtu.be/d3kd5KbGB48)

- [Hash Tables and Dictionaries](https://youtu.be/sfWyugl4JWA)

**## Recommended**

- [C++ POINTERS FULL COURSE Beginner to Advanced (150min)](https://youtu.be/kiUGf\_Z08RQ)

ACTIVITY1

**# Activities**

**## Task 1**

- Refer to the following link. Discuss how bubble sort works:

<https://opendsa-server.cs.vt.edu/embed/bubblesortAV>

**## Task 2**

- Refer to the following link. Your task is to show the behavior for one iteration of the outer for loop of Bubble Sort (Try at least 3 cases).

<https://opendsa-server.cs.vt.edu/ODSA/Exercises/Sorting/BubsortPRO.html>

**## Task 3**

- The following snippet is from `./src/bubble.cpp` lines 16-28. Discuss in groups how the code works:

```cpp

    for (i = 0; i < 10; i++)

    {

        for (j = i + 1; j < 10; j++)

        {

            if (a[j] < a[i])

            {

                temp = a[i];

                a[i] = a[j];

                a[j] = temp;

            }

        }

        pass++;

    }

```

- The following snippet is from `./src/selection.cpp` lines 34-41. Discuss in groups how the code works:

```cpp

    for (j = i + 1; j < 10; j++)

    {

        if (myarray[j] < ele\_small)

        {

            ele\_small = myarray[j];

            position = j;

        }

    }

```

**## Task 4: Individual, at home**

- Discuss the complexity analysis of selection sort. Refer to the link below:

<https://www.softwaretestinghelp.com/selection-sort/>

**## Links**

- <https://cpp.sh/>

ANSWERS:

**## Task 1**

- Refer to the following link. Discuss how bubble sort works:

<https://opendsa-server.cs.vt.edu/embed/bubblesortAV>

Graphical user interface, application

Description automatically generated

Bubble sort is a sorting algorithm that works by repeatedly stepping through lists that need to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. This passing procedure is repeated until no swaps are required, indicating that the list is sorted

**## Task 2**

- Refer to the following link. Your task is to show the behavior for one iteration of the outer for loop of Bubble Sort (Try at least 3 cases).

<https://opendsa-server.cs.vt.edu/ODSA/Exercises/Sorting/BubsortPRO.html>

Move from left to right through the array.

**At each position, if the value is greater than the value to its right, then swap them.**

Graphical user interface, text, application, email

Description automatically generated

**## Task 3**

- The following snippet is from `./src/bubble.cpp` lines 16-28. Discuss in groups how the code works:

```cpp

    for (i = 0; i < 10; i++)

    {

        for (j = i + 1; j < 10; j++)

        {

            if (a[j] < a[i])

            {

                temp = a[i];

                a[i] = a[j];

                a[j] = temp;

            }

        }

        pass++;

    }

```

- The following snippet is from `./src/selection.cpp` lines 34-41. Discuss in groups how the code works:

```cpp

    for (j = i + 1; j < 10; j++)

    {

        if (myarray[j] < ele\_small)

        {

            ele\_small = myarray[j];

            position = j;

        }

    }

```

ANSWERS

#include <iostream>

using namespace std;

int main()

{

    int i, j, temp, pass = 0;

    int a[10] = {10, 2, 0, 14, 43, 25, 18, 1, 5, 45};

    cout << "Input list ...\n";

    for (i = 0; i < 10; i++)

    {

        cout << a[i] << "\t";

    }

    cout << endl;

    for (i = 0; i < 10; i++)

    {

        for (j = i + 1; j < 10; j++)

        {

            if (a[j] < a[i])

            {

                temp = a[i];

                a[i] = a[j];

                a[j] = temp;

            }

        }

        pass++;

    }

    cout << "Sorted Element List ...\n";

    for (i = 0; i < 10; i++)

    {

        cout << a[i] << "\t";

    }

    cout << "\nNumber of passes taken to sort the list:" << pass << endl;

    return 0;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture3-main\activity1\src> .\bubble

Input list ...

10 2 0 14 43 25 18 1 5 45

Sorted Element List ...

0 1 2 5 10 14 18 25 43 45

Number of passes taken to sort the list:10

#include <iostream>

using namespace std;

int findSmallest(int[], int);

int main()

{

    int myarray[10] = {11, 5, 2, 20, 42, 53, 23, 34, 101, 22};

    int pos, temp, pass = 0;

    cout << "\n Input list of elements to be Sorted\n";

    for (int i = 0; i < 10; i++)

    {

        cout << myarray[i] << "\t";

    }

    for (int i = 0; i < 10; i++)

    {

        pos = findSmallest(myarray, i);

        temp = myarray[i];

        myarray[i] = myarray[pos];

        myarray[pos] = temp;

        pass++;

    }

    cout << "\n Sorted list of elements is\n";

    for (int i = 0; i < 10; i++)

    {

        cout << myarray[i] << "\t";

    }

    cout << "\nNumber of passes required to sort the array: " << pass;

    return 0;

}

int findSmallest(int myarray[], int i)

{

    int ele\_small, position, j;

    ele\_small = myarray[i];

    position = i;

    for (j = i + 1; j < 10; j++)

    {

        if (myarray[j] < ele\_small)

        {

            ele\_small = myarray[j];

            position = j;

        }

    }

    return position;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture3-main\activity1\src> .\selection

Input list of elements to be Sorted

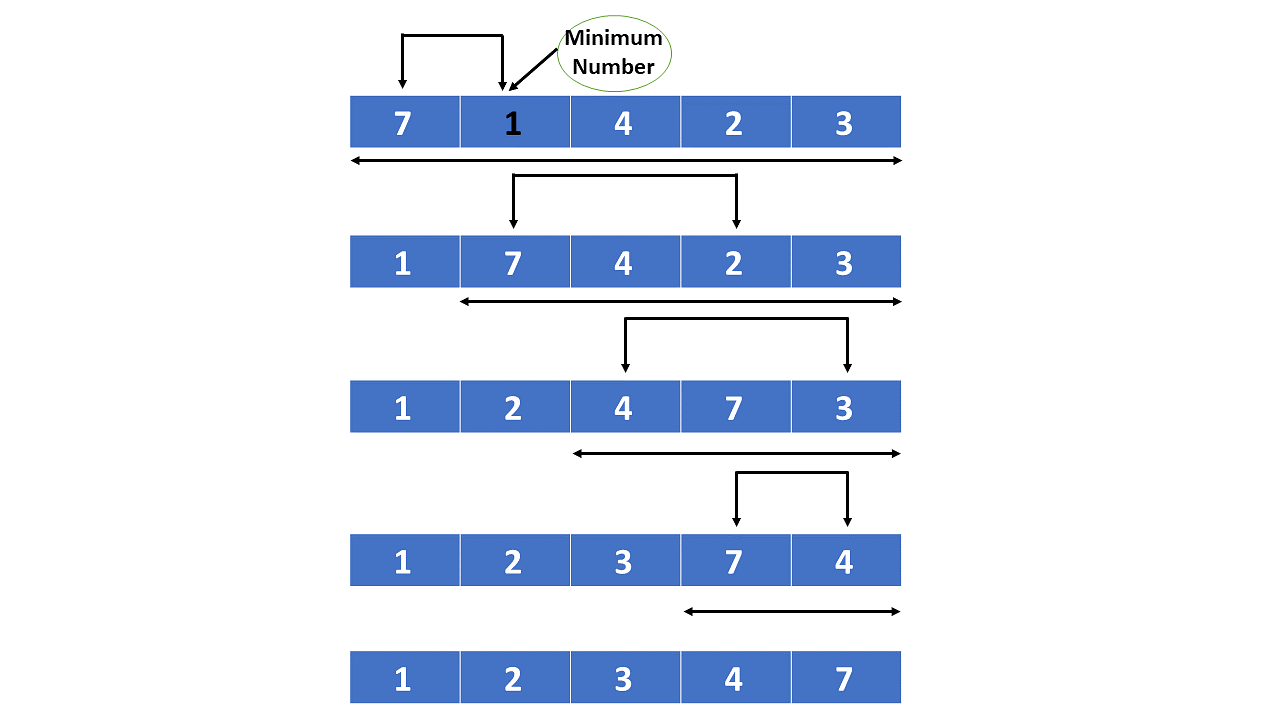
11 5 2 20 42 53 23 34 101 22

Sorted list of elements is

2 5 11 20 22 23 34 42 53 101

Number of passes required to sort the array: 10

Selection sort is an effective and efficient sort algorithm based on comparison operations. It adds one element in each iteration. You need to select the smallest element in the array and move it to the beginning of the array by swapping with the front element.



**## Task 4: Individual, at home**

- Discuss the complexity analysis of selection sort. Refer to the link below:

<https://www.softwaretestinghelp.com/selection-sort/>

**## Links**

- <https://cpp.sh/>

**An In-Depth Look At Selection Sort In C++ With Examples.**

As the name itself suggests, the selection sort technique first selects the smallest element in the array and swaps it with the first element in the array.

Next, it swaps the second smallest element in the array with the second element and so on. Thus for every pass, the smallest element in the array is selected and put in its proper position until the entire array is sorted.

**Introduction**

Selection sort is quite a straightforward sorting technique as the technique only involves finding the smallest element in every pass and placing it in the correct position.

Selection sort works efficiently when the list to be sorted is of small size but its performance is affected badly as the list to be sorted grows in size.

Hence we can say that selection sort is not advisable for larger lists of data.

**General Algorithm**

The General Algorithm for selection sort is given below:

**Selection Sort (A, N)**

**Step 1**: Repeat Steps 2 and 3 for K = 1 to N-1  
**Step 2**: Call routine smallest(A, K, N,POS)  
**Step 3**: Swap A[K] with A [POS]  
[End of loop]  
**Step 4**: EXIT

**Routine smallest (A, K, N, POS)**

* **Step 1**: [initialize] set smallestElem = A[K]
* **Step 2**: [initialize] set POS = K
* **Step 3**: for J = K+1 to N -1,repeat  
  if smallestElem > A [J]  
  set smallestElem = A [J]  
  set POS = J  
  [if end]  
  [End of loop]
* **Step 4**: return POS

**Pseudocode For Selection Sort**

|  |
| --- |
| Procedure selection\_sort(array,N)      array – array of items to be sorted      N – size of array  begin  **for** I = 1 to N-1      begin          set min  = i  **for** j = i+1 to N          begin  **if** array[j] < array[min] then                  min = j;              end **if**          end **for**          //swap the minimum element with current element  **if** minIndex != I then              swap array[min[] and array[i]          end **if**      end **for**  end procedure |

An example to illustrate this selection sort algorithm is shown below.

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\Programs> .\selection\_sort

Input list of elements to be Sorted

11 5 2 20 42 53 23 34 101 22

Sorted list of elements is

2 5 11 20 22 23 34 42 53 101

Number of passes required to sort the array: 10

**Complexity Analysis Of Selection Sort**

As seen in the pseudocode above for selection sort, we know that selection sort requires two for loops nested with each other to complete itself. One for loop steps through all the elements in the array and we find the minimum element index using another for loop which is nested inside the outer for loop.

Therefore, given a size N of the input array, the selection sort algorithm has the following time and complexity values.

|  |  |
| --- | --- |
| Worst case time complexity | O( n 2 ) ; O(n) swaps |
| Best case time complexity | O( n 2 ) ; O(n) swaps |
| Average time complexity | O( n 2 ) ; O(n) swaps |
| Space complexity | O(1) |

The time complexity of O(n2) is mainly because of the use of two for loops. Note that the selection sort technique never takes more than O(n) swaps and is beneficial when the memory write operation proves to be costly.

### Conclusion

Selection sort is yet another simplest sorting technique that can be easily implemented. Selection sort works best when the range of the values to be sorted is known. Thus as far as sorting of data structures using selection sort is concerned, we can only sort data structure which are linear and of finite size.

This means that we can efficiently sort data structures like arrays using the selection sort.

In this tutorial, we have discussed selection sort in detail including the implementation of selection sort using C++ and Java languages. The logic behind the selection sort is to find the smallest element in the list repeatedly and place it in the proper position.

In the next tutorial, we will learn in detail about insertion sort which is said to be a more efficient technique than the other two techniques that we have discussed so far i.e. bubble sort and selection sort.

ACTIVITY 2

**# Activities**

**## Task 1**

- Refer to the following link. Discuss how Insertion sort works:

<https://opendsa-server.cs.vt.edu/ODSA/AV/Sorting/insertionsortAV.html>

**## Task 2**

- The following snippet is from `./src/insertion.cpp` lines 12-22. Discuss in groups how the code works:

```cpp

    for (int k = 1; k < 10; k++)

    {

        int temp = myarray[k];

        int j = k - 1;

        while (j >= 0 && temp <= myarray[j])

        {

            myarray[j + 1] = myarray[j];

            j = j - 1;

        }

        myarray[j + 1] = temp;

    }

```

**## Task 3**

- Discuss the complexity analysis of insertion sort. Refer to the link below:

<https://www.softwaretestinghelp.com/insertion-sort/>

**## Links**

- <https://cpp.sh/>

ANSWERS:

**## Task 1**

- Refer to the following link. Discuss how Insertion sort works:

<https://opendsa-server.cs.vt.edu/ODSA/AV/Sorting/insertionsortAV.html>

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time by comparisons. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.

Graphical user interface, application

Description automatically generated



**Pseudo Code**

procedure insertionSort(A: list of sortable items)

n = length(A)

for i = 1 to n - 1 do

j = i

while j > 0 and A[j-1] > A[j] do

swap(A[j], A[j-1])

j = j - 1

end while

end for

end procedure

// C++ program for insertion sort

#include <bits/stdc++.h>

using namespace std;

// Function to sort an array using

// insertion sort

void insertionSort(int arr[], int n)

{

    int i, key, j;

    for (i = 1; i < n; i++)

    {

        key = arr[i];

        j = i - 1;

        // Move elements of arr[0..i-1],

        // that are greater than key, to one

        // position ahead of their

        // current position

        while (j >= 0 && arr[j] > key)

        {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

// A utility function to print an array

// of size n

void printArray(int arr[], int n)

{

    int i;

    for (i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\Programs> .\insertion\_sort

5 6 11 12 13