BENCHMARKING TOOL FOR SORTING AND SEARCHING ALGORITHMS

1701246, Necip Gözüaçık
1806392, Okan Büyüktepe
Course: CMP5151 Software Design Patterns
Lecturer: Ph.D. Emre Kaplan

AGENDA

- DESCRIPTION
- FEATURES
- DESIGN PATTERNS
- RESULTS
- REFERENCES

DESCRIPTION

- This project aims to run **sorting** and **searching** algorithms and provide some statistics such as execution time and applied number of steps. These statistics help us to make a comparison between the algorithms from the point of time complexity level.
- The project calculates execution time results and applied number of steps per sorting and searching with using built-in array structures corresponding to use cases (**Best, Worst and Average**).
- User can be able to display the performance results as a figure for the selected algorithm type.

FEATURES

- Following Core features are contracted about
  - *Supported Sorting Algorithms:* ***Selection, Bubble, Merge, Quick***
  - *Supported Searching Algorithms:* ***Linear, Binary, Jump, Interpolation***
  - *Supported Complexity Use Cases:* ***Best, Average, Worst***
  - *Complexity level (Best/Average/Worst) and Algorithm (Sorting/Searching) can be selected from GUI.*
  - *Comparison results are displayed as plot on GUI.*

DESIGN PATTERNS

- Following design patterns are used during implementation. Python programming language with QT framework is used.
  - *Simple Factory*
  - *Iterator*
  - *Template*
  - *Visitor*

DESIGN PATTERNS

- Simple Factory

```
class CaseFactory():
    def buildCase(self,type):
        if type == "Best":
            return BestCase()
        elif type =="Worst":
            return WorstCase()
        elif type == "Average":
            return AverageCase()
```

- Iterator

```
class SortIterator(object):
    "An iterator."

    def __init__(self, container):
        self.container = container
        self.n = -1

    def __next__(self):
        self.n += 1
        if self.n >= self.container.sortMax:
            raise StopIteration
        return self.container.sortTypes[self.n],self.n

    def __iter__(self):
        return self
```

DESIGN PATTERNS

- Template

```
class PerformanceSort(Common.Performance):
    def __init__(self,title):
        self.timeStart = 0
        self.timeStop = 0
        self.title = title
        self.elapsedTime = 0
        self.numberofSteps = 0


    def startTime(self):
        #tempTime = datetime.time(datetime.now())
        self.timeStart =  timer()
        #self.timeStart = (tempTime.second*1000) +
(tempTime.microsecond/1000)
        # self.timeStart =
float(datetime.time(datetime.now()).microsecond)
        print("Start Time: ",datetime.datetime.now())


    def sort(self,arr):
        return list

    def measurePerformance(self,arr):
        self.startTime()
        self.sort(arr)
        self.stopTime()
        self.calcTime()
        return self.elapsedTime, self.numberofSteps
```

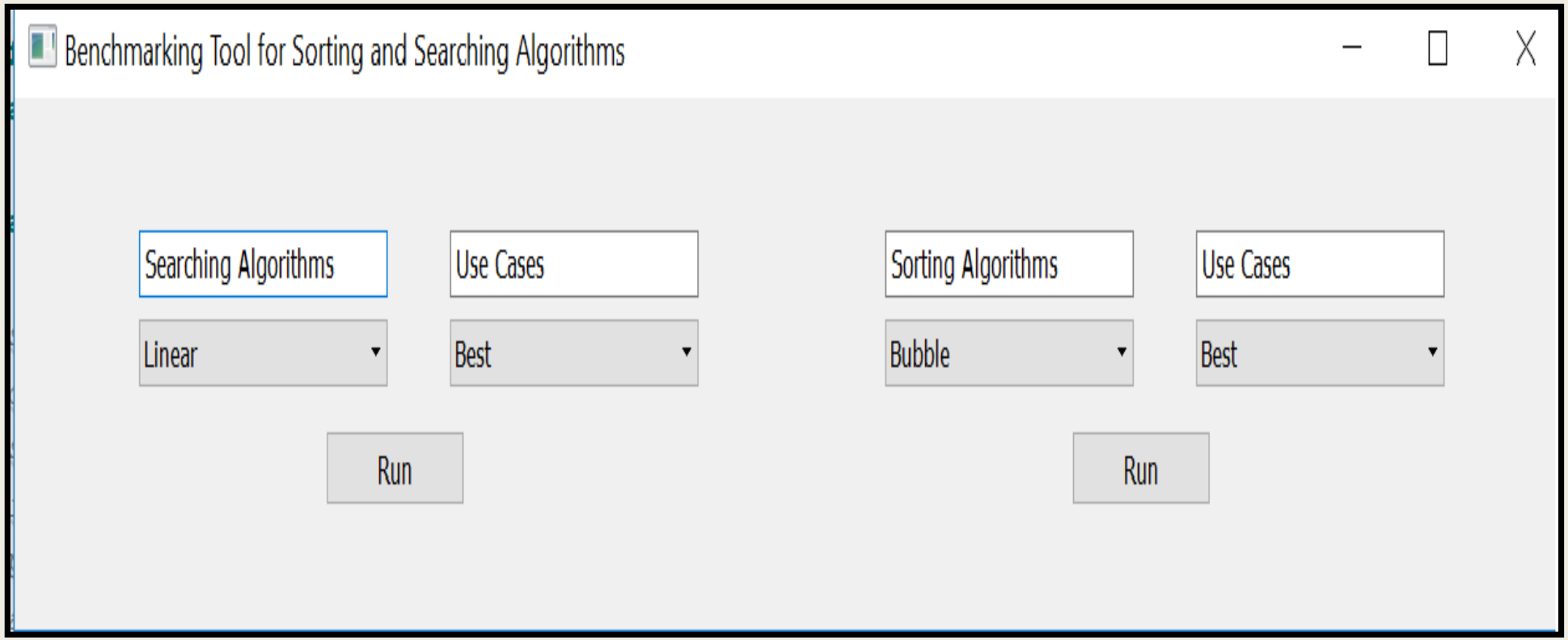
```
class BubbleSort(PerformanceSort):
    def sort(self,arr):

#start_time=float(datetime.time(datetime.now()).microsecond)
        #print(start_time)
        n = len(arr)
        for i in range(n):
            self.numberofSteps = self.numberofSteps + 1
            swapped = False
            for j in range(0, n - i - 1):
                self.numberofSteps = self.numberofSteps + 1
                if arr[j] > arr[j + 1]:
                    arr[j], arr[j + 1] = arr[j + 1], arr[j]
                    swapped = True
            if swapped == False:
                break
        # stop_time =
float(datetime.time(datetime.now()).microsecond)
        # print(stop_time)
        # self.elapsedTime = float(stop_time - start_time)
        return arr
```

DESIGN PATTERNS

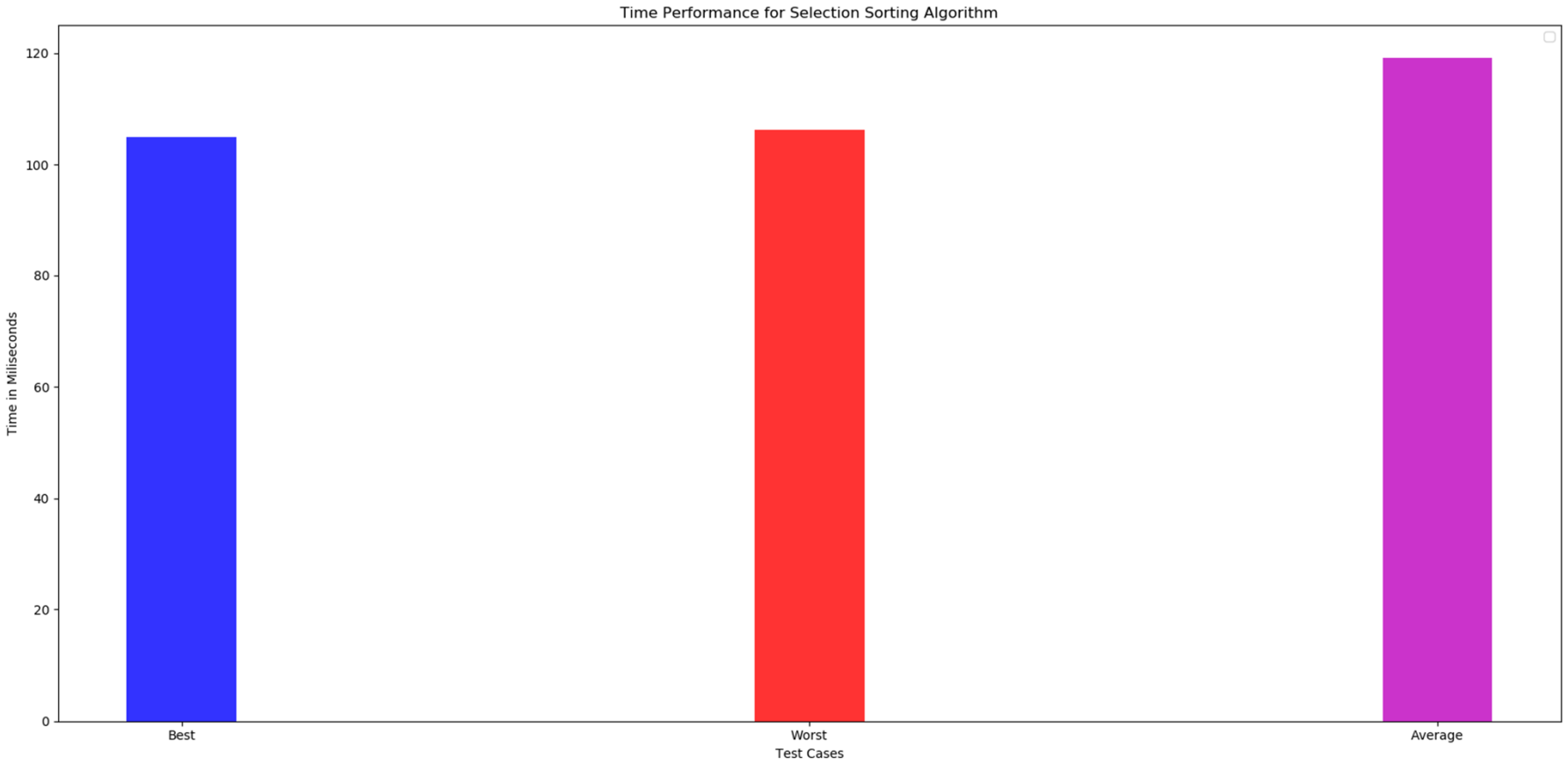
- Visitor

```
class Visitor():
    def visitSearch(self):pass
    def visitSort(self):pass

class SearchVisit(object):
    def accept(self, visitor):
        visitor.visitSearch()

class SortVisit(object):
    def accept(self, visitor):
        visitor.visitSort()

class AlgVisitor(Visitor):
    def __init__(self,type,case):
        self.type = type
        self.case = case

    def visitSearch(self):
        print("1")
        if self.type == 'All':
            Search.runAll(self.type, self.case)
        else:
            Search.runSearch(self.type, self.case)

    def visitSort(self):
        print("2")
        if self.type == 'All':
            Sorting.runAll(self.type, self.case)
        else:
            Sorting.runSort(self.type, self.case)
```

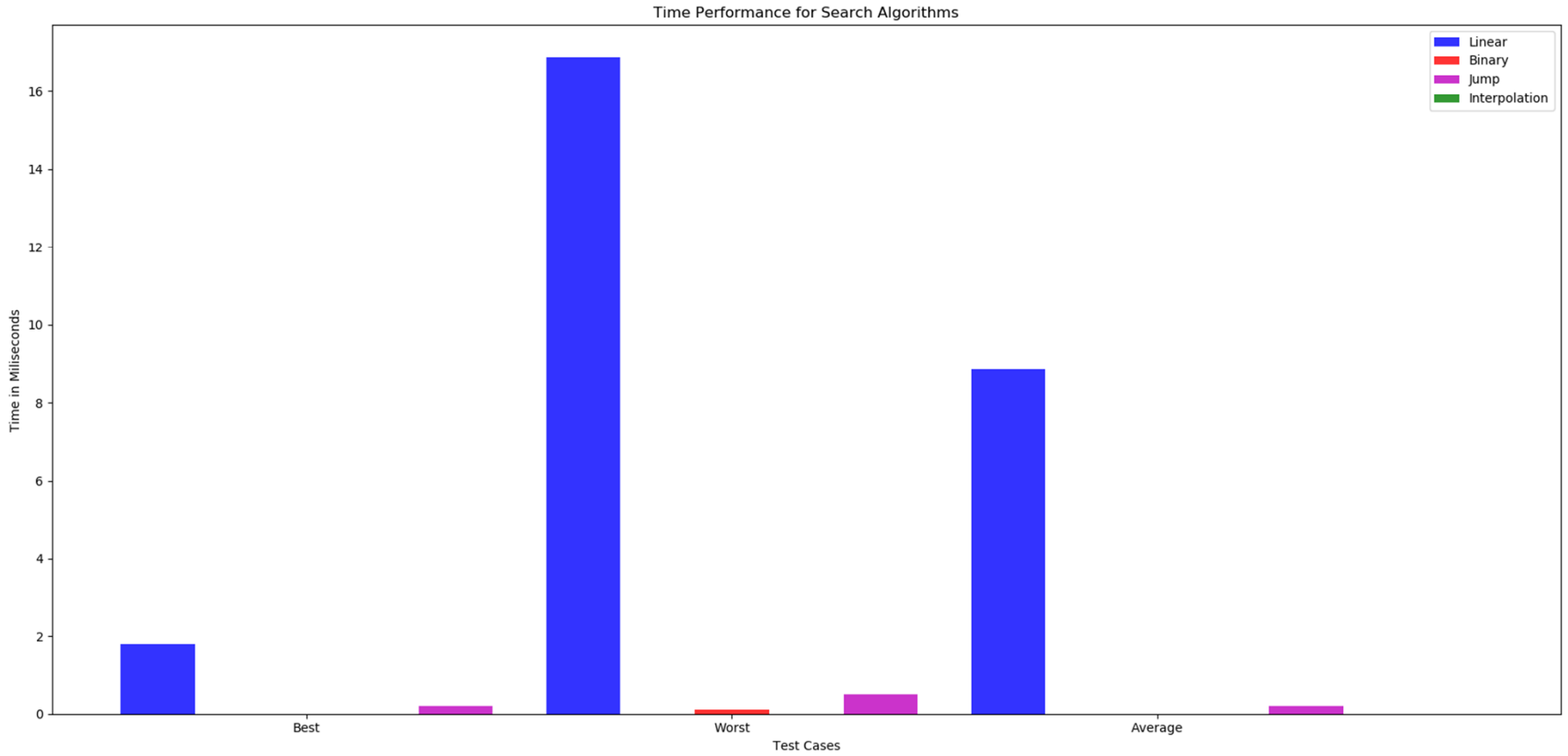
RESULTS


Benchmarking Tool for Sorting and Searching Algorithms
Searching Algorithms
Use Cases
Linear
Best
Run
Sorting Algorithms
Use Cases
Bubble
Best
Run

RESULTS


Time Performance for Selection Sorting Algorithm
Time in Miliseconds
0
20
40
60
80
100
120
Best
Worst
Average
Test Cases

RESULTS


Time Performance for Search Algorithms
Linear
Binary
Jump
Interpolation
Time in Miliseconds
0
2
4
6
8
10
12
14
16
Best
Worst
Average
Test Cases

REFERENCES

[1] https://www.geeksforgeeks.org/linear-search/

[2] https://www.geeksforgeeks.org/binary-search/

[3] https://www.geeksforgeeks.org/jump-search/

[4] https://www.geeksforgeeks.org/interpolation-search/

[5] https://www.geeksforgeeks.org/selection-sort/

[6] https://www.geeksforgeeks.org/quick-sort/

[7] https://www.geeksforgeeks.org/merge-sort/

[8] https://www.geeksforgeeks.org/bubble-sort/

[9} https://github.com/gozuacik/cmp5151


Thank you!