Lab 3

Implementing Binary Heap & Sorting Techniques

Name	ID
Muhammad El Kotb	19016258
Ahmed Adel Abudef	19015264

Table Of Contents

1. Problem Statement

- **Max-Heap**

In this assignment, you're required to implement some basic procedures and show how they

could be used in a sorting algorithm:

The MAX-HEAPIFY procedure, which runs in O(lg n) time, is the key to maintaining

the max-heap property.

The BUILD-MAX-HEAP procedure, which runs in linear time, produces a max-heap from

an unordered input array.

• The HEAPSORT procedure, which runs in O(n lg n) time, sorts an array in place.

The MAX-HEAP-INSERT, and HEAP-EXTRACT-MAX procedures, which run in O(lg n) time, allow the heap data structure to implement a priority queue.

- **Sorting Techniques**

You’re required to compare the running time performance of your algorithms against:

– An O(n2) sorting algorithm such as Selection Sort, Bubble Sort, or Insertion sort.

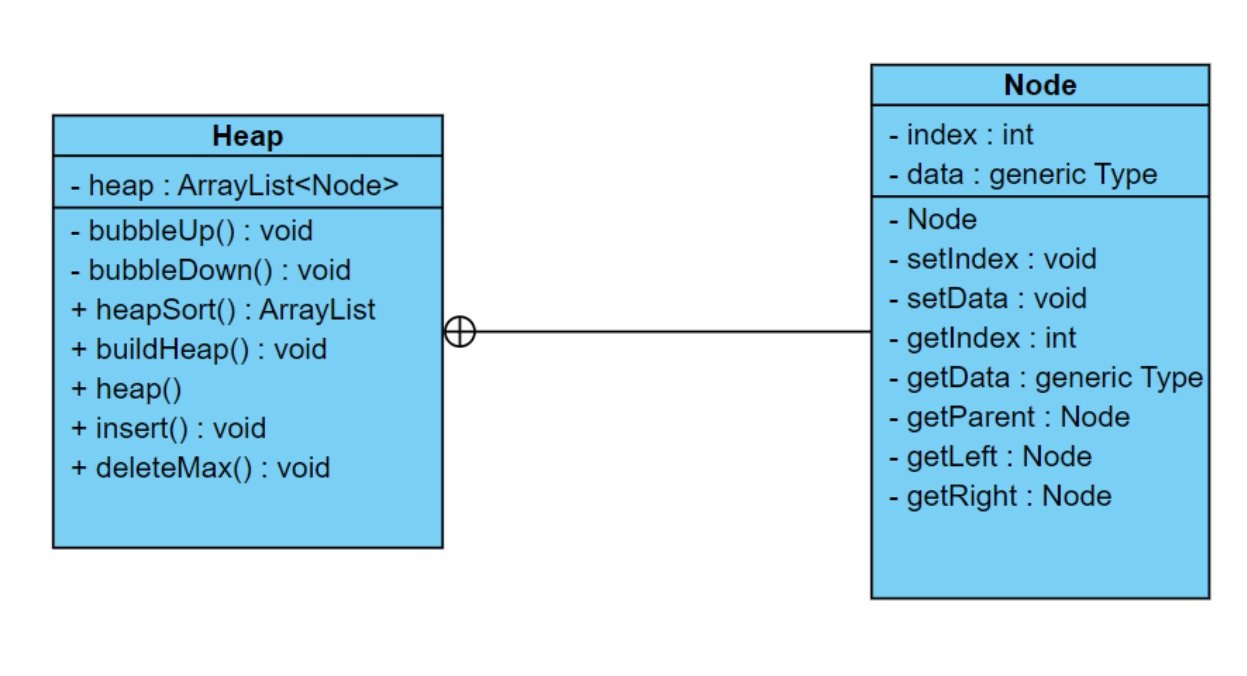
– An O(n lg n) sorting algorithm such as Merge Sort or Quick sort algorithm in the average case.

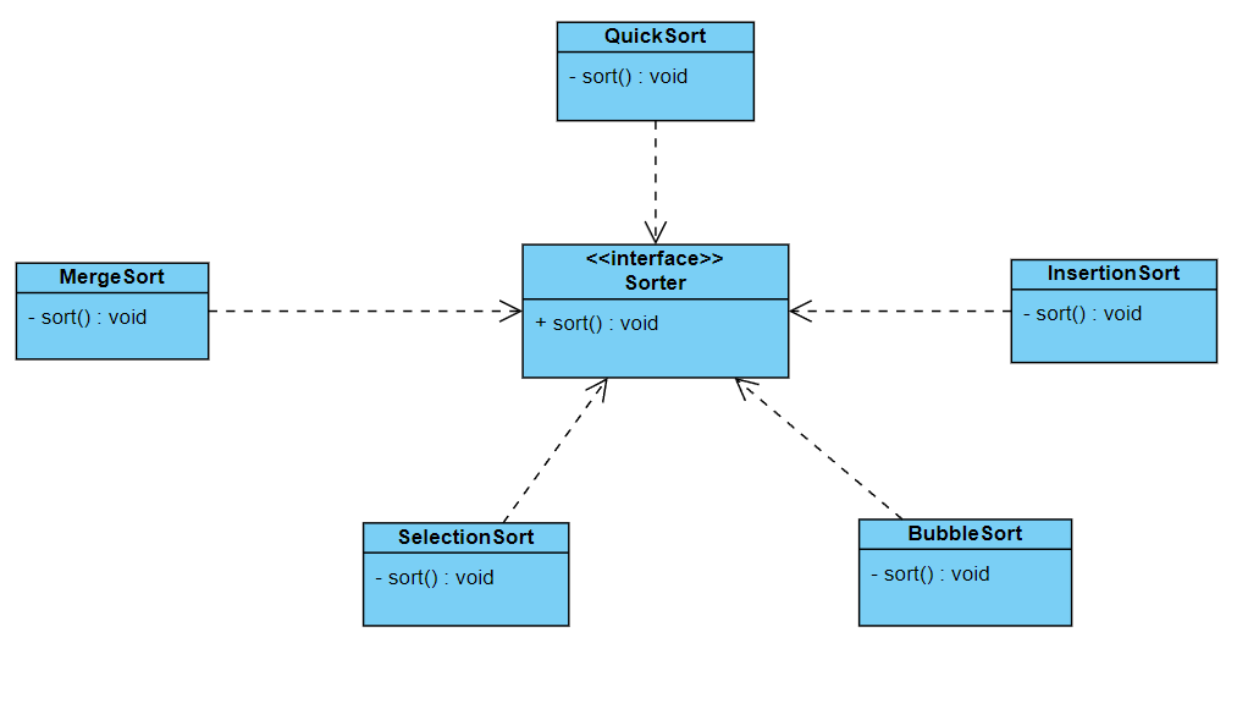
You’re required to compare the running time performance of your algorithms against:

– An O(n2) sorting algorithm such as Selection Sort, Bubble Sort, or Insertion sort.

– An O(n lg n) sorting algorithm such as Merge Sort or Quick sort algorithm in the average case.

2.UML Diagram





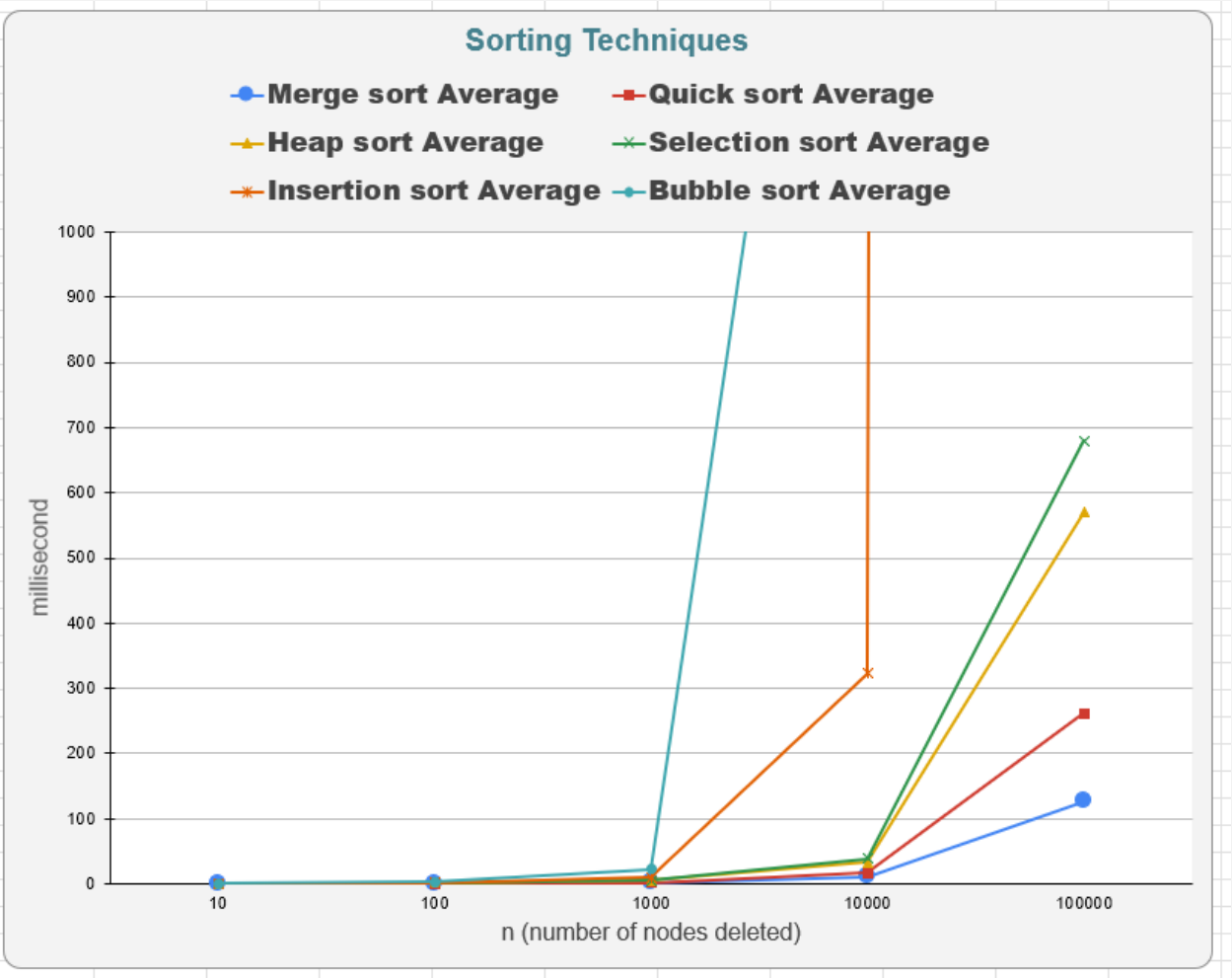
3.Analysis

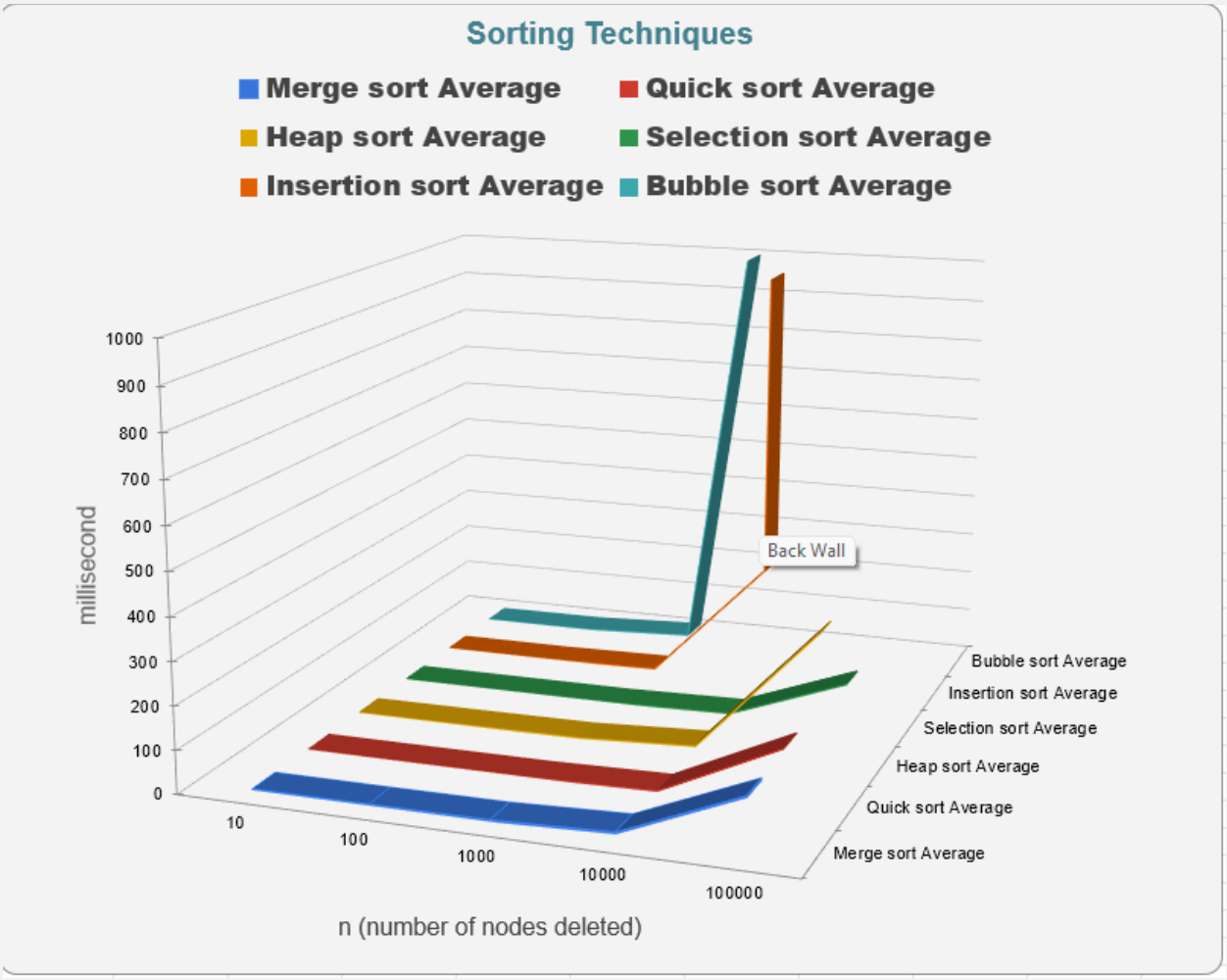
- Analysis was performed on 5 batches (10, 100, 1000, 10000, 100000) nodes.
- The time for every batch of nodes was estimated 10 times and the average value was calculated at the end.

Analysis Table:

n					Bubble sort					Bubble sort Average	
10	0	0	0	0	0	0	0	0	0	0	0
100	1	2	1	2	1	2	1	1	0	1	1.2
1000	19	11	12	12	11	11	12	12	14	17	13.1
10000	1617	1779	2047	1948	1897	1664	2787	2410	1765	1463	1937.7
100000											
n					Insertion sort					Insertion sort Average	
10	0	0	0	0	0	0	0	0	0	0	0
100	0	0	0	1	0	0	0	0	0	0	0.1
1000	7	4	5	4	4	4	4	4	4	3	4.3
10000	268	272	281	348	258	278	310	285	281	256	283.7
100000	107288	73201	107783	106759	110581	112101	132558	118718	119727	116728	110544.4
n					Selection sort					Selection sort Average	
10	0	0	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0	0	0
1000	0	1	0	0	1	0	1	0	1	1	0.5
10000	7	6	6	6	7	6	6	6	7	7	6.4
100000	112	135	94	106	107	103	118	114	100	100	108.9
n					Merge sort					Merge sort Average	
10	0	0	0	0	0	0	0	0	0	0	0
100	0	1	0	0	1	0	0	0	1	0	0.3
1000	2	2	1	2	1	1	1	1	1	2	1.4
10000	23	20	14	9	8	8	7	7	6	6	10.8
100000	138	140	105	98	166	110	136	135	121	112	126.1
n					Quick sort					Quick sort Average	
10	0	0	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0	0	0
1000	1	0	1	0	1	0	1	0	1	0	0.5
10000	6	6	5	5	6	5	5	5	5	6	5.4
100000	142	174	179	142	117	108	113	118	123	136	135.2
n					Heap sort					Heap sort Average	
10	3	0	1	0	0	0	1	0	1	1	0.7
100	3	2	1	2	1	1	1	0	0	1	1.2
1000	4	4	3	2	2	3	4	3	2	2	2.9
10000	26	17	15	13	16	16	13	14	12	19	16.1
100000	284	431	564	293	256	256	274	226	259	247	309

Charts:


Sorting Techniques
Merge sort Average
Quick sort Average
Heap sort Average
Selection sort Average
Insertion sort Average
Bubble sort Average
1000
900
800
700
600
500
400
300
200
100
0
millisecond
10
100
1000
10000
100000
n (number of nodes deleted)


Sorting Techniques
Merge sort Average
Quick sort Average
Heap sort Average
Selection sort Average
Insertion sort Average
Bubble sort Average
millisecond
1000
900
800
700
600
500
400
300
200
100
0
Back Wall
10
100
1000
10000
100000
n (number of nodes deleted)
Bubble sort Average
Insertion sort Average
Selection sort Average
Heap sort Average
Quick sort Average
Merge sort Average