

SCHOOL OF COMPUTER SCIENCE

UNIVERSITI SAINS MALAYSIA

CPT212 : Design and Analysis of Algorithm

Semester 2, Academic Session: 2019/2020

Lecturer : Dr. Teh Je Sen

|  |  |  |
| --- | --- | --- |
| Name | Matric Number | Task Division |
| Chan Siang Sheng | 142413 |  |
| Tang Wen Shuen | 142623 |  |

Table of Content

|  |  |  |
| --- | --- | --- |
| No | Content | Page |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1.0 Problem requirement

Problem:

An input of list consisting of only integer in ascending order. The integer can be in negative and positive, thus making the list ascending from negative integers to positive integers. The integer will be randomly generated from -10 to 10 in the list. Print a list of squared of the input list in an ascending order.

Input and Constraint:

*n* = input size of list

*ar* = list

-10 *≤ ar [i] ≤* 10

Output:

A list of squared of the input list in ascending order.

Sample Input:

*n* =10

*ar* = [-7, -6, -3, -2, 0, 1, 4, 5, 6, 8]

Expected Sample Output:

[0, 1, 4, 9, 16, 25, 36, 36, 49, 64]

Explanation:

1. 0 is the smallest integer after squaring 0 from the input list.

[0]

1. 1 is the second smallest integer after squaring 1 from the input list.

[0, 1]

1. 4 is the next smallest integer after squaring -2 from the input list.

[0, 1, 4]

1. 16 is the next smallest integer after squaring -3 from the input list.

[0, 1, 4, 9]

1. 16 is the next smallest integer after squaring 4 from the input list.

[0, 1, 4, 9, 16]

1. This repetition continues until 64 which is the square of 8 that is greatest in the input list.

[0, 1, 4, 9, 16, …, 64]

2.0 Solution Approach

In this problem, we have identified 2 algorithms to solve this question. One of the algorithm is O(n) and the other is O(n log n). Both algorithms are written in Python 3 and programmed in .py format. The explanation of both algorithms will be done with the following sample input:

* *n* = 10
* *ar* = [-7, -6, -3, -2, 0, 1, 4, 5, 6, 8]

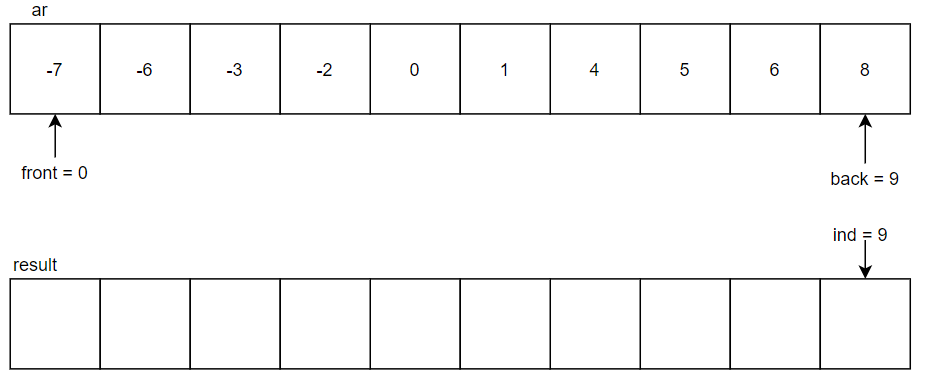
2.1 Algorithm of Big-O(*n*)

This algorithm will be assisted with 3 variables as well as 1 list for output as stated below:

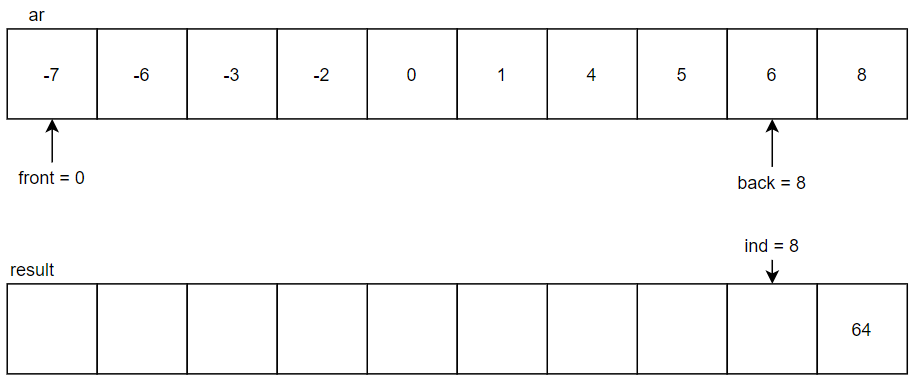
* *front*, an integer variable that point at the index of first element from the input list that has yet to be square
* *back*, an integer variable that point at the index of last element from the input list that has yet to be square
* *ind*, an integer variable that point at the last index of output list that to be inserted by integer after squaring it.
* *result*, an output list with same input size or length as input list

Steps:

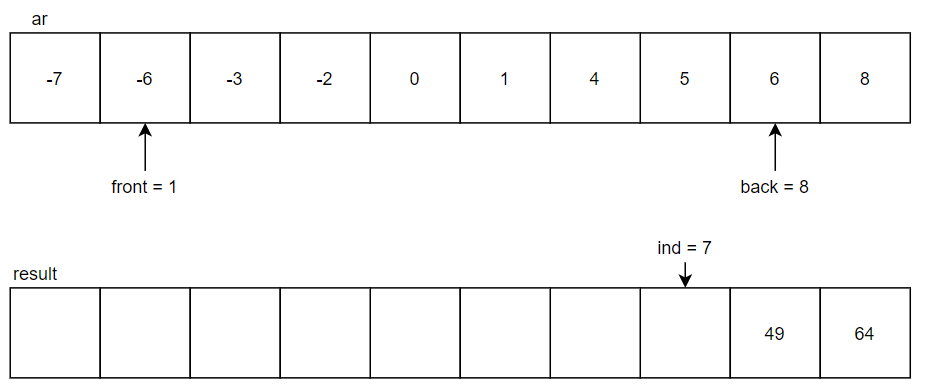
1. The initial state of algorithm before entering first cycle of while loop is shown as below:



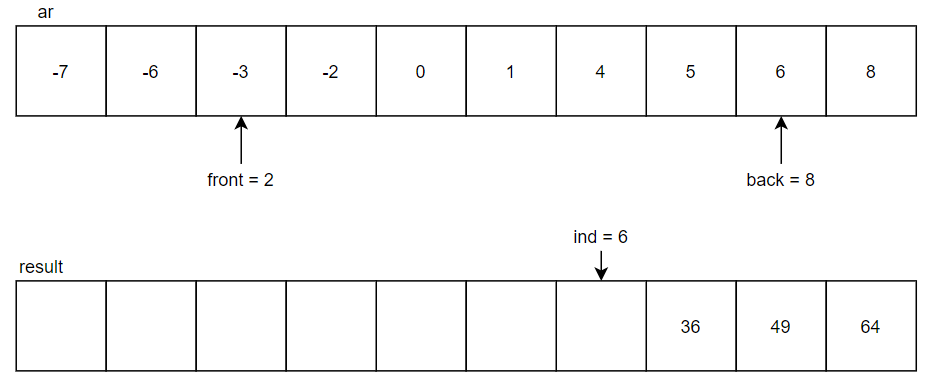
1. In the 1st cycle of while loop, -7 in the index 0 will be taken with absolute and compares with 8 in the index 9. Since 8 is greater than 7, 8 will be squared to 64 and put into the index 9 of the result list. The ***back*** will be decremented to point at index 8 and ***ind*** will be decremented to point at index 8 while there is no change to ***front***.



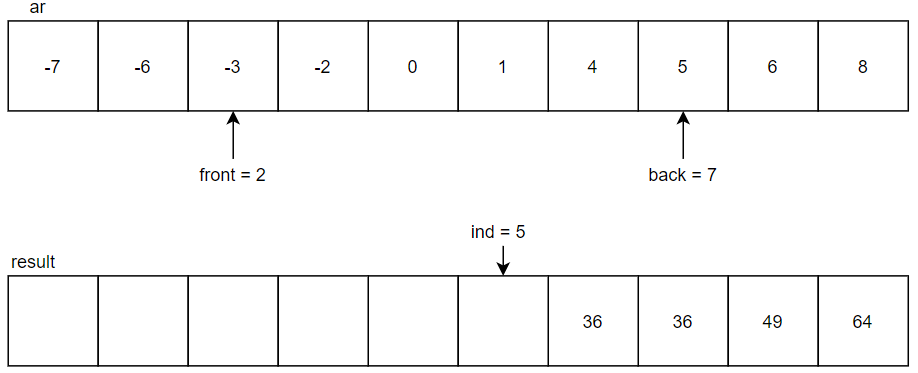
1. In the 2nd cycle, -7 in the index 0 will be taken with absolute and compares with 6 in the index 8. Since 7 is greater than 6, 7 will be squared to 49 and put into the index 8 of the result list. The ***front*** will be incremented to point at index 1 and ***ind*** will be decremented to point at index 7, while ***back*** will remain no change.



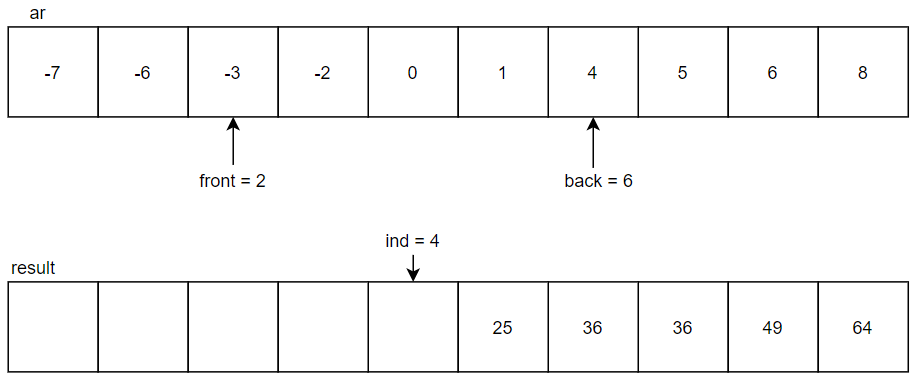
1. In the next cycle, -6 in the index 1 will be absolute and compares with 6 in the index 8. Since both are 6, 6 in index 1 will be squared to 36 and put into the index 7 of the result list. The***front*** will be incremented to point at index 2 and ***ind*** will be decremented to point at index 6.



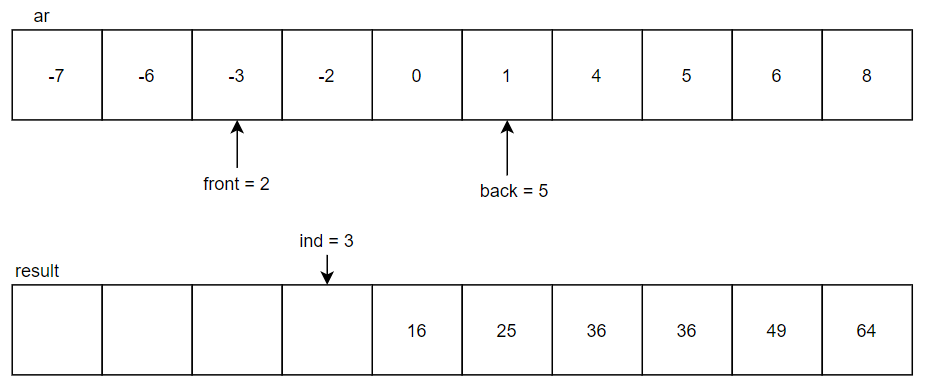
1. In the next cycle, -3 in the index 2 will be absolute and compares with 6 in the index 8. Since 6 is greater than 3, 6 will be squared to 36 and put into the index 6 of the result list. The ***back*** will be decremented to point at index 7 and ***ind*** will be decremented to point at index.



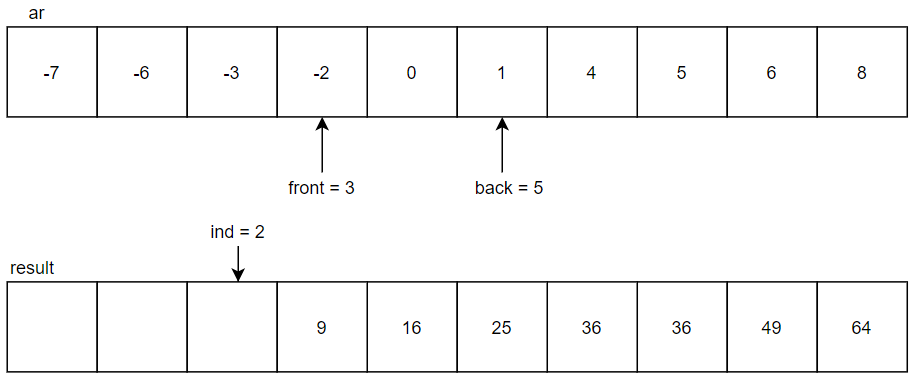
1. In the next cycle, -3 in the index 2 will be absolute and compares with 5 in the index 7. Since 5 is greater than 3, 5 will be squared to 25 and put into the index 5 of the result list. The ***back*** will be decremented to point at index 6 and ***ind*** will be decremented to point at index 4.



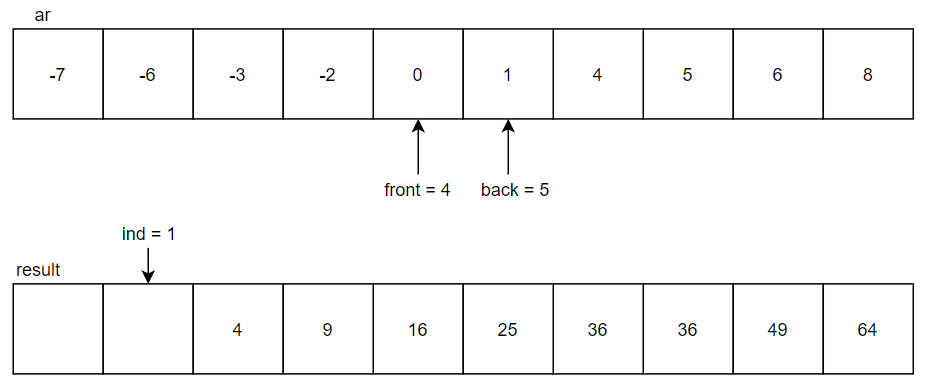
1. In the next cycle, -3 in the index 2 will be absolute and compares with 4 in the index 6. Since 4 is greater than 3, 4 will be squared to 16 and put into the index 4 of the result list. The ***back*** will be decremented to point at index 5 and***ind*** will be decremented to point at index 3.



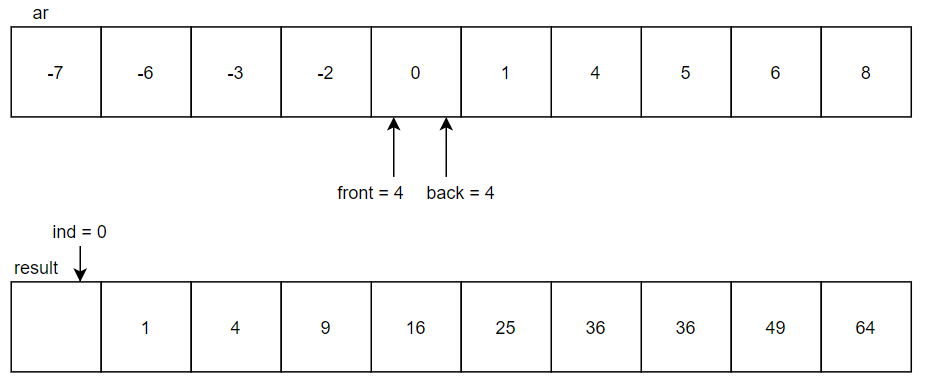
1. In the next cycle, -3 in the index 2 will be absolute and compares with 1 in the index 5. Since 3 is greater than 1, 3 will be squared to 9 and put into the index 3 of the result list. The***front*** will be incremented to point at index 3 and ***ind*** will be decremented to point at index 2.



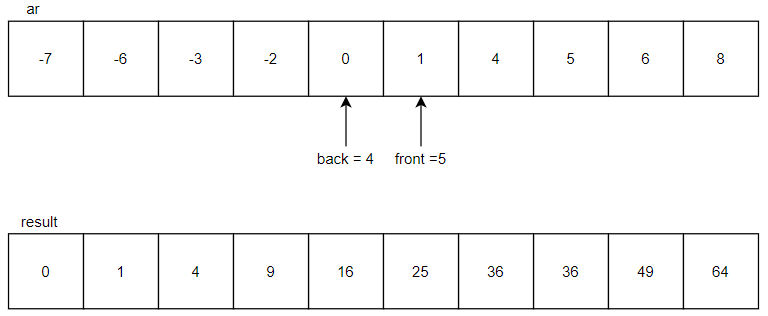
1. In the next cycle, -2 in the index 3 will be absolute and compares with 1 in the index 5. Since 2 is greater than 1, 2 will be squared to 4 and put into the index 2 of the result list. The ***front*** will be incremented to point at index 4 and ***ind*** will be decremented to point at index 1.



1. In the next cycle, 0 in the index 4 will be compared with 1 in the index 5. Since 1 is greater than 0, 1 will be squared to 1 and put into the index 1 of the result list. The ***back*** will be decremented to point at index 4 and***ind*** will be decremented to point at index 0.



1. In the next cycle, 0 in the index 4 will be compared with 0 in the index 4. Since both are 0, 0 will be squared to 0 and put into the index 0 of the result list. The ***front*** will be incremented to point at index 5 and ***ind*** will be decremented to point at index -1. When ***ind*** is lower than 0, the while loop ends at here with 10th time of cycles. As such, the result list is then completed.



2.2 Pseudocode of Big-O(*n*)

*n* ← input size

*ar* ← list with random number between -10 to 10

*ar* ← sorting into ascending order

function bign (*ar*)

*front* ← 0

*back* ← size of list -1

*ind* ← size of list -1

**while** *ind* ≤ 0

**if** abs(*ar*[*front*]) ≥ *ar*[*back*] **then**

*ans*[*ind*] ← ar[*front*])

*front* ← *front* +1

**else if** abs(*ar*[*front*]) < *ar*[*back*] **then**

*ans*[*ind*] ← ar[*back*])

*back* ← back -1

*ind* ← *ind* -1

**print** *ans*

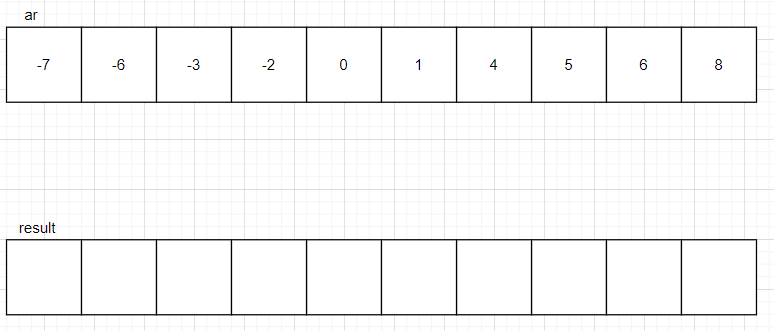
2.3 Algorithm of Big-O(*n log n*)

This algorithm only required 1 list for output as stated below:

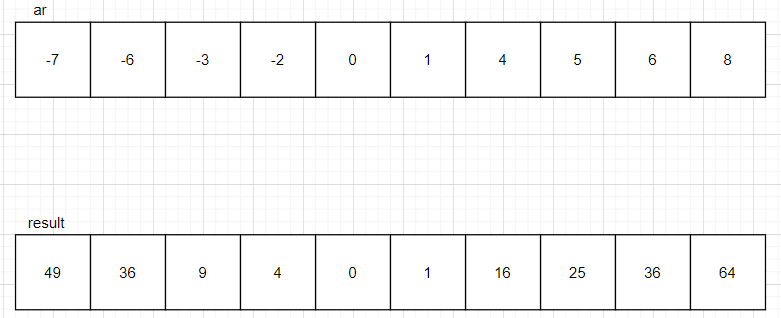
* *result*, an output list with same input size or length as input list

Steps:

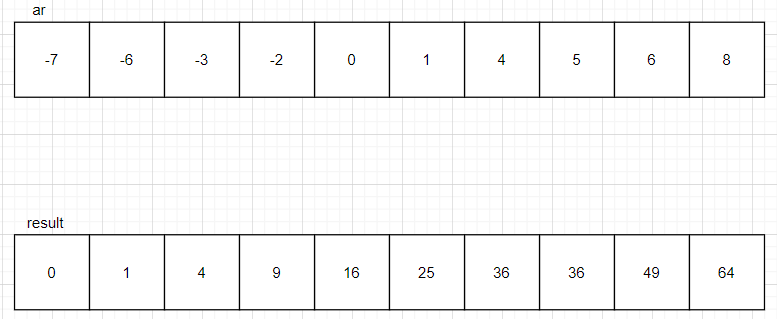
1. The initial state of algorithm before entering first cycle of for loop is shown as below:



1. In the for loop, each element of the list will be squared and assigned continuously into the result list. After each element is squared and inserted into the *result* list, the for loop ends here.



1. After the end of for loop, the *result* list will be sorted in ascending order. And the *result* list is ready to be printed.



2.3 Pseudocode of Big-O(*n log n*)

*n* ← input size

*ar* ← list with random number between -10 to 10

*ar* ← sorting into ascending order

function bignlogn (*ar*)

**for** *i* ← 0, …, *n*-1 **do**

*ans[i]* ← *ar[i]*\* *ar[i]*

*ans* ← sorting into ascending order

**print** *ans*