Name:.....

Student ID:.....

Assignment # 1: Arrays

1. Describe the definition of an array.

Answer:

An array is a chunk of memory, consisting of equal-size elements. Each of those elements have an integer index, which uniquely refers to the value stored. The values are all of the same type (integer, character, etc.).

2. Given a 1-D array:
   Start Address is 1000,
   element size is 4,

What is the address of the element at index 10? Also, provide the calculation.

Answer:

Using this formula: start_address + elem_size * index = address
We get: 1000 + 4(10) = 1040

Address	1000	1004	1008	1012	1016	1020	1024	1028	1032	1036	1040
Index	0	1	2	3	4	5	6	7	8	9	10

3. Given a 1-D array:
   Start Address is 2000,
   address of the element at index 20 is 2160,

What is the element size of this array? Also, provide the calculation.

Answer:

Using this formula: start_address + elem_size * index = address
We get: 2000 + elem_size(20) = 2160
Therefore, elem_size = (2160 - 2000) / 20 = 8

4. Given a 2-D array of rank 3 of size 4:
   Start Address is 3000,
   element size is 4,

What is the address of the element at index (2,3)?
Also, provide the calculation.
Reminder: rank index starts at 0.

Answer:

Using this formula: start_address + elem_size * index = address

Since this array is of rank 3, size 4, we can calculate index by this formula:
index(i,j) = index(i * size_of_rank + j)

For index (2,3), we get: Start_address + Elem_size * ((2 * 4 + (3))
= 3000 + (4 * 11) = 3044

5. Given an array called 'data' below:

Value	**9**	**10**	**5**	**7**	**3**	**1**	**0**	**2**
Index	0	1	2	3	4	5	6	7

5.1 Write the Python operations used and asymptotic running time *O*(?) executed for each operation to obtain the array below.

9	**10**	**5**	**7**	**3**	**1**		

Answer:

Remove 2 at the end of an array: data.pop() = *O*(1)
Remove 0 at the end of an array: data.pop() = *O*(1)

5.2 Write the Python operations used and asymptotic running time *O*(?) executed for each operation to obtain the array below.

9	10	7	3	1			

Answer:

Remove 5 in between an array: data.remove(5) or data.pop(2) = *O*(n)

5.3 Write the Python operations used and asymptotic running time *O*(?) executed for each operation to obtain the array below.

12	10	7	4	8	3	1	

Answer:

Change 9 to 12 at the beginning of an array: data[0] = 12 = *O*(1)
Insert 4 in between an array: data.insert(3, 4) = *O*(n)
Insert 8 in between an array: data.insert(4, 8) = *O*(n)

5.4 Write the Python operations used and asymptotic running time *O*(?) executed for each operation to obtain the array below.

1	3	8	4	7	10	12	

Answer:

Reverse array/list: data.reverse() = *O*(n)

5.5 Write the Python operations used and asymptotic running time O(?) executed for each operation to obtain the array below.

1	3	4	7	8	10	12	

Answer:

Sort array/list: data.sort() = *O*(n log n)

6. Consider a case study, suppose Facebook database keeps a list of usernames. When a user tries to log into Facebook website, a search is queried into the database looking for the entered username. If their username is in the list of usernames, that user can log in.

6.1 In this case, can you use an array to store the list of usernames? And why?

Answer:

We can't use a fixed-size array for storing a list of usernames because usernames are strings and they can have different lengths. Each element of the array uses the same number of bytes.

However, we can use a referential-type array, such as Python list, that stores the address of the usernames since the address has fixed-sized bytes (e.g. 8 bytes). This allows constant-time access for any username in the list.

7. Tic-tac-toe is a game played on a 3-by-3 board. This can be represented by a 2-D array of rank 3, size 3. Two players 'X' and 'O' take turns in placing their marks in the cells on the board. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row or column is the winner.

   A 2-D array of tic-tac-toe call 'board' is as follows:

Index	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)
2	(2,0)	(2,1)	(2,2)

   Write all winning conditions for this game by using 2-D array elements.

For example, board[0][0] == board[0][1] == board[0][2].

```
board[0][0] == board[0][1] == board[0][2]        # row 0
board[1][0] == board[1][1] == board[1][2]        # row 1
board[2][0] == board[2][1] == board[2][2]        # row 2
board[0][0] == board[1][0] == board[2][0]        # column 0
board[0][1] == board[1][1] == board[2][1]        # column 1
board[0][2] == board[1][2] == board[2][2]        # column 2
board[0][0] == board[1][1] == board[2][2]        # diagonal
board[0][2] == board[1][1] == board[2][0]        # reverse diagonal
```

8. Using Python or pseudocode, write an algorithm for binary search for a sorted sequence.
   - Parameter low and high should be initialised first outside the loop.
   - Think carefully what should be the condition for the while loop to stop executing (Hint: think about what should occur when low and high have the same index).
   - For if-else condition, it should compare whether target value is less than or more than the data[mid], and then assign new high or low accordingly.

```
# Iterative version
def binary_search_iterative(data, target):
  """ Return True if target is found in the given Python
list."""
  low = 0
  high = len(data)-1
  while low <= high:
    mid = (low + high) // 2
    if target == data[mid]:
      return True
    elif target < data[mid]:
      # only consider the left portion left of the middle
      high = mid - 1
    else:
      # only consider the right portion of the middle
      low = mid + 1
 return False
```