Lab Experiment No. 8

Objective

Implementation and analysis of Dynamic Arrays and its operations.

- Insertion
- Deletion
- Searching
- Traversing
- Updation

Theory

A dynamic array is a data structure that allows elements to be added or removed and resizes itself automatically when capacity is exceeded. Unlike static arrays with fixed size, dynamic arrays grow or shrink as needed. They provide random access like arrays and are useful when the number of elements is not known in advance.

Operations typically supported by dynamic arrays:

- Insertion: Adding a new element, possibly resizing the array.
- Deletion: Removing an element and shifting others.
- Searching: Finding an element by value or index.
- Traversing: Visiting all elements.
- Updation: Changing the value at a specific index.

Algorithm:

1. Insertion

1. Check if the current size equals capacity.
2. If yes, double the capacity and reallocate memory.
3. Insert the new element at the current size index.
4. Increment the size.

2. Deletion

1. Check if index is valid.
2. Shift all elements after the index to the left by one.
3. Decrement the size.

4. Optionally, shrink the capacity if size is too small.

3. Searching

1. Iterate over the array.
2. If the element matches the target, return the index.
3. If not found, return -1.

4. Traversing

1. Loop from 0 to size-1.
2. Print each element.

5. Updation

1. Check if index is valid.
2. Replace the element at the index with the new value.

Code

T-5.1. Implementation using Arrays. Depth-First Search (DFS)

```
#include <iostream>
using namespace std;

class DynamicArray {
private:
    int* arr;
    int size;
    int capacity;

    void resize(int newCapacity) {
        int* temp = new int[newCapacity];
        for (int i = 0; i < size; ++i)
            temp[i] = arr[i];
        delete[] arr;
        arr = temp;
        capacity = newCapacity;
    }

public:
    DynamicArray() {
        size = 0;
        capacity = 2;
        arr = new int[capacity];
    }

    void insert(int value) {
        if (size == capacity)
    int search(int value) {
        for (int i = 0; i < size; ++i) {
            if (arr[i] == value)
                return i;
        }
        return -1;
    }

    void update(int index, int value) {
        if (index < 0 || index >= size) {
            cout << "Invalid index\n";
            return;
        }
        cout << "Updated index " << index << " from "
<< arr[index] << " to " << value << "\n";
        arr[index] = value;
    }

    void traverse() {
        cout << "Array Elements: ";
        for (int i = 0; i < size; ++i)
            cout << arr[i] << " ";
        cout << "\n";
    }

    ~DynamicArray() {
        delete[] arr;
```

```
        resize(capacity * 2);
    arr[size++] = value;
    cout << "Inserted: " << value << "\n";
}

 void deleteAt(int index) {
    if (index < 0 || index >= size) {
        cout << "Invalid index\n";
        return;
    }
    cout << "Deleted: " << arr[index] << "\n";
    for (int i = index; i < size - 1; ++i)
        arr[i] = arr[i + 1];
    size--;
    if (size < capacity / 2 && capacity > 2)
        resize(capacity / 2);
}
```

```
    }
};

int main() {
    DynamicArray da;

    da.insert(10);
    da.insert(20);
    da.insert(30);
    da.traverse();

    da.deleteAt(1);
    da.traverse();

    int index = da.search(30);
    if (index != -1)
        cout << "Element 30 found at index " << index <<
"\n";
    else
        cout << "Element 30 not found\n";

    da.update(0, 99);
    da.traverse();

    return 0;
}
```

Sample Output

```
Inserted: 10
Inserted: 20
Inserted: 30
Array Elements: 10 20 30
Deleted: 20
Array Elements: 10 30
Element 30 found at index 1
Updated index 0 from 10 to 99
Array Elements: 99 30
```

Complexity Analysis

Table 8.1 Analysis of time complexity for Dynamic Array.

Operation	Best Case	Worst Case	Average Case
Insertion	O(1) (at end, no resize)	O(n) (with shifting/resizing)	O(n) (in middle or beginning)

Deletion	O(1) (from end)	O(n) (with shifting)	O(n) (from middle or beginning)
Searching	O(1) (if element at known index)	O(n) (linear search)	O(n) (not found)
Traversal	O(n)	O(n)	O(n)

Conclusion

Dynamic Arrays are a flexible alternative to static arrays with the ability to resize dynamically. They support efficient insertion and updates, although deletion and searching remain linear-time operations. This experiment demonstrates core dynamic array operations and their performance characteristics compared to basic static structures.