# Vectors in C++

→ Array leads to wastage of space.
→ " needs memory reallocation.
→ Vectors, Rich library function
→ Easy to know size.
→ by default initialized with default value.

Vector programs →

declaration → vector<int> v; → empty

v.push_back(10); → 10

v.push_back(6); → 10 6

size of vectors ← v.size()

→ Element can be accessed like arrays. v[i]. or v.at(i)

array index out of bounds checking.

RANKA
DATE / /
PAGE

## Another Method →

```
vector<int> v {10, 5, 20};
for (int x : v)
    cout << x << " ";
```

↳ Goes to every element of vector

O/P = 10 5 20

```
→ for (int &x : v)
      x = 6
  print vector.
```

if it is reference nothing changes.

↓

print = 6 6 6

## Another way for creations

```
int n = 3, x = 10
vector<int> v(n, x);
```

→ 10 10 10

```
for (auto it = v.begin(); it != v.end(); it++)
    cout << (*it) << " "
```

↙ Iterators

## Another way →

```
int n = sizeof(arr)/sizeof(arr[0]);
vector<int> v(arr, arr+n);
```

v.rbegin() → last element

v.rend() → first element.

→ c.begin() → is constant version of previous iterator

RANKA
DATE / /
PAGE

**Best Way**

```
vector <int> v {10, 5, 20};

for (auto x : v)
    cout << x << " ";
return 0;
```

→ for vector traversal.

More functions in vector →

1. v.pop_back() → used to remove last element of vector.
2. v.front() → Reference to first element O(1)
3. v.back() → — — last element O(1)
4. v.insert() → O(n)

vector <int> v {10, 5, 20, 15}

* auto it = v.insert(v.begin(), 100);
  {100, 10, 5, 20, 15}
* v.insert(v.begin()+2, 200}
  {100, 10, 200, 5, 20, 15}
* v.insert(v.begin, 2, 300} → 300, 2 times.
  {300, 300, 100, 10, 200, 5, 20, 15}
* vector <int> v2;
  v2.insert(v2.begin(), v.begin(), v.begin()+2)
  → {300, 300}

Till position last but not included

* v.size → O(1)
* begin(), rbegin(), crbegin() etc. → O(1)

O(n)

(5) v.erase() :→

* v.erase(v.begin()) → only first element
* v.erase(v.begin(), v.end()-1) → Remove all

(6) v.clear() → Remove all elements of vector

(7) v.empty() → Tell vector contains any element/not → O(1)

(8) v.resize() → Resize the function (empty)

v.resize(3) → Reduce size by 3.

v.resize(8, 100) → default value for remaining empty spaces becomes 100.

O(n)

## Time complexities

push_back → worst case O(n), avg O(1)

pop_back → O(n), avg O(1)

* vector is not modified unless it is not passed by reference.
→ (vector<int>&v)

Key Points (How vector works) -

→ internally use dynamically allocated arrays only.

→ if current allowed space becomes full do following :-

(a) Create a new layer space of double size

(b) copy elements from old space to new

(c) Free old space.

# Keeping Index after Sorting

I/P → arr[] = {20, 40, 30, 10}

O/P →

| array elements | Index |
|---|---|
| 10 | 3 |
| 20 | 0 |
| 30 | 2 |
| 40 | 1 |

→ Sort and mention their original index

→ Idea is to use vector of pairs.

```
void printSortedWithIndex(int arr[], int n)
{
vector<pair<int,int>> v;
for(int i=0; i<n; i++)
    v.push_back({arr[i], i});
sort(v.begin(), v.end());
for(auto x : v)
    cout<<x.first<<" "<<x.second<<endl;
}
```