MCA - Week 4: Arrays
Name: Suraj Vishwakarma
Qid: 24510031

1. Insert an element at a given index in an array

```
 #include <bits/stdc++.h>
using namespace std;
vector<int> insertAt(vector<int> a, int val, int
    idx){ int n = a.size();
    if(idx < 0 || idx > n) return a; // invalid index: return unchanged
    a.push_back(0); // increase size
    for(int i = n; i > idx; --i) a[i] =
    a[i-1]; a[idx] = val;
    return a;
}
```

2. Delete an element from an array

```
#include <bits/stdc++.h>
using namespace std;
vector<int> deleteAt(vector<int> a, int idx){
    int n = a.size();
    if(idx < 0 || idx >= n) return a; // invalid
    for(int i = idx; i+1 < n; ++i) a[i] = a[i+1];
    a.pop_back();
    return a;
}
```

3. Linear Search in an array

```
#include <bits/stdc++.h>
using namespace std;
int linearSearch(const vector<int>& a, int key){
    for(int i = 0; i < (int)a.size(); ++i)
        if(a[i] == key) return i;
    return -1;
}
```

4. Binary Search (iterative and recursive)

```
#include <bits/stdc++.h>
using namespace std;
int binaryIter(const vector<int>& a, int
    key){ int l = 0, r = (int)a.size()-1;
    while(l <= r){
        int m = l + (r-l)/2;
        if(a[m] == key) return m;
        if(a[m] < key) l = m+1; else r = m-1;
    }
    return -1;
}
int binaryRec(const vector<int>& a, int l, int r, int key){
    if(l>r) return -1;
    int m = l + (r-l)/2;
    if(a[m]==key) return m;
    if(a[m] < key) return binaryRec(a, m+1, r, key);
    else return binaryRec(a, l, m-1, key);
}
int main(){
    vector<int> a = {1,2,5,7,9};
    cout<<binaryIter(a,7)<<" "<<binaryRec(a,0,a.size()-1,5)<<"\n";
}
```

5. Find maximum and minimum element in an array

```
 #include <bits/stdc++.h>
using namespace std;
pair<int,int> minMax(const vector<int>& a){
    if(a.empty()) return {INT_MAX,
    INT_MIN}; int mn = a[0], mx = a[0];
    for(int x: a){ mn = min(mn,x); mx = max(mx,x); }
    return {mn,mx};
}
```

6. Find the second largest element in an array

```
#include <bits/stdc++.h>
using namespace std;
int secondLargest(const vector<int>&
    a){ int n = a.size();
    if(n < 2) return INT_MIN; // not
    defined int first = INT_MIN, second =
    INT_MIN; for(int x: a){
        if(x > first){ second = first; first = x;}
        else if(x > second && x < first) second = x;
    }
    return second==INT_MIN ? INT_MIN : second;
```

7. Reverse an array in-place using two-pointer technique

```
// C++: reverse in-place using two pointers
#include <bits/stdc++.h>
using namespace std;
void reverseArr(vector<int>& a){
    int l = 0, r =
    (int)a.size()-1;
    while(l < r){ swap(a[l++], a[r--]); }
}
```

8. Maximum sum of k consecutive elements (sliding window)

```
// C++: max sum of k consecutive elements
#include <bits/stdc++.h>
using namespace std;
long long maxSumK(const vector<int>& a, int
    k){ int n = a.size();
    if(k>n) return LLONG_MIN;
    long long sum = 0;
    for(int i=0;i<k;i++) sum += a[i];
    long long best = sum;
    for(int i=k;i<n;i++){
        sum += a[i] - a[i-k];
        best = max(best,
        sum);
    }
    return best;
}
int main(){
    vector<int> a = {1,2,3,4,5};
    cout<<maxSumK(a,2)<<"\n"; // 9 (4+5)
}
```

Explanation: Time: O(n), Space: O(1).

9. Two Sum (find two numbers sum equals target)

```
// C++: Two-sum (returns indices) using hash map
#include <bits/stdc++.h>
using namespace std;
pair<int,int> twoSum(const vector<int>& a, int
    target){ unordered_map<int,int> mp; // value ->
    index for(int i=0;i<(int)a.size();++i){
        int need = target - a[i];
        if(mp.count(need)) return {mp[need], i};
        mp[a[i]] = i;
    }
    return {-1,-1};
}
int main(){
    vector<int> a = {2,7,11,15};
```

```
    auto p = twoSum(a,9);
    cout<<p.first<<' '<<p.second<<"\n"; // 0 1
}
```

Explanation: Time: O(n), Space: O(n).

10. Maximum subarray sum using Kadane's algorithm

```
// C++: Kadane's algorithm
#include <bits/stdc++.h>
using namespace std;
long long kadane(const vector<int>& a){
    long long maxEnding = a[0], maxSoFar = a[0];
    for(size_t i=1;i<a.size();++i){
        maxEnding = max<long long>(a[i], maxEnding + a[i]);
        maxSoFar = max(maxSoFar, maxEnding);
    }
    return maxSoFar;
}
int main(){
    vector<int> a = {-2,1,-3,4,-1,2,1,-5,4};
    cout<<kadane(a)<<"\n"; // 6 (subarray 4,-1,2,1)
}
```

Explanation: Time: O(n), Space: O(1).

SECTION B: Objective Questions

1. Answer: (b) — Arrays are stored in contiguous memory blocks.
2. Answer: (b) — Access by index is O(1).
3. Answer: (a) — Linear search checks elements sequentially: O(n).
4. Answer: (a) — Binary search requires sorted array.
5. Answer: (a) — Binary search complexity is O(log n).
6. Answer: (a) — Appending at end (if capacity available) is O(1).
7. Answer: (b) — Inserting in middle needs shifting: O(n).
8. Answer: (b) — Deletion from middle needs shifting: O(n).
9. Answer: (b) — Traversal visits all elements: O(n).
10. Answer: (a) — Best case linear search when found at first index: O(1).
11. Answer: (a) — Worst case linear search is O(n).
12. Answer: (a) — Arrays require O(1) extra memory (in-place).
13. Answer: (b) — Index range: 0 to n-1.
14. Answer: (b) — Two-pointer technique used for pair-sum, etc. (answer b)
15. Answer: (a) — Sliding window is used for subarray problems.
16. Answer: (a) — Kadane runs in linear time O(n).
17. Answer: (a) — Kadane uses constant extra space O(1).
18. Answer: (b) — Removing duplicates from sorted array is O(n).
19. Answer: (b) — Naive rotation can be O(k*n).
20. Answer: (a) — Reversal algorithm rotates in O(n).
21. Answer: (a) — Maximum subarray solved by Kadane's algorithm.
22. Answer: (d) — Linked list is a different data structure (not an application of arrays).
23. Answer: (a) — Binary search recurrence: T(n)=T(n/2)+O(1).
24. Answer: (b) — Merging two sorted arrays of sizes m and n is O(m+n).

25. Answer: (c) — Majority element can be solved in O(n) (Boyer-Moore).
26. Answer: (a) — Searching in unsorted array needs linear search.
27. Answer: (c) — Access is the fastest (O(1)).
28. Answer: (b) — Array is a non-primitive data structure.
29. Answer: (c) — Two-sum uses two-pointer technique when sorted.
30. Answer: (a) — Fixed-size arrays are static.

SECTION C:

1. Array Leaders

```
// C++: Leaders in an array - elements greater than all elements to their right
#include <bits/stdc++.h>
using namespace std;
vector<int> leaders(const vector<int>& a){
    int n = a.size();
    vector<int> res;
    int mx = INT_MIN;
    for(int i = n-1; i>=0; --i){
        if(a[i] > mx){ res.push_back(a[i]); mx = a[i];}
    }
    reverse(res.begin(), res.end());
    return res;
}
int main(){
    vector<int> a = {16,17,4,3,5,2};
    for(int x: leaders(a)) cout<<x<<' '; // 17 5 2
}
```

2. Sort 0's,1's,2's — Approach: Dutch National Flag algorithm (low, mid, high pointers).

```
// C++: Sort 0s,1s,2s - Dutch National Flag
(one-pass) #include <bits/stdc++.h>
using namespace std;
void sort012(vector<int>& a){
    int low=0, mid=0, high=(int)a.size()-1;
    while(mid <= high){
        if(a[mid]==0) swap(a[low++],
        a[mid++]); else if(a[mid]==1) mid++;
        else swap(a[mid], a[high--]);
    }
}
int main(){
    vector<int> a =
    {2,0,2,1,1,0}; sort012(a);
    for(int x: a) cout<<x<<' ';
}
```