DSA Cheatsheet

Arrays & Strings

```
Declaration: int arr[10]; | vector<int> v(n);
Traversal: for (int i : arr) or for (int i = 0; i < n; i++)
Patterns: Sliding Window, Prefix Sum, Two Pointers
Prefix Sum: prefix[i] = prefix[i-1] + arr[i];
Two Pointers: i = 0, j = n-1; while(i < j) { ... }
```

Sorting Algorithms

```
Bubble: O(n^2), Insertion: O(n^2), Merge: O(n log n), Quick: O(n log n)
In-place Quick Sort:
void qs(int a[], int l, int r) {
  if (l >= r) return;
  int p = a[r], i = l;
  for (int j = l; j < r; j++) if (a[j] <= p) swap(a[i++], a[j]);
  swap(a[i], a[r]); qs(a, l, i-1); qs(a, i+1, r);
}
```

Trees

```
Inorder: left -> root -> right
Preorder: root -> left -> right
Postorder: left -> right -> root
Height: max(height(l), height(r)) + 1
BST: Left < Root < Right, operations O(log n)
```

Hash Maps & Sets

```
unordered_map<int,int> m; unordered_set<int> s;
Insert/Search O(1) avg. Frequency: for (int x : arr) m[x]++;
```

Stacks & Queues

```
Stack: stack<int> s; (LIFO)
Queue: queue<int> q; (FIFO)
Min Stack: push(val, min_so_far)
```

Recursion & Backtracking

```
void backtrack(vector<int>& sol) {
  if (done) output;
  for (...) if (valid) { sol.push_back(...); backtrack(sol); sol.pop_back(); }
}
```

Dynamic Programming

```
Steps: define dp[i], recurrence, base case
Fibonacci:
dp[0]=0; dp[1]=1; for(i=2;i<=n;i++) dp[i]=dp[i-1]+dp[i-2];
```

Graphs

```
Adjacency List: vector<vector<int>> g(n);
```

DSA Cheatsheet

DFS:

```
void dfs(int u) { vis[u] = 1; for (int v : g[u]) if (!vis[v]) dfs(v); }
```

BFS, Dijkstra (PQ), MST, Topo Sort

Searching

Binary Search:

```
int bs(vector<int>& a, int t) {
  int l=0, r=a.size()-1;
  while(l<=r) { int m=(l+r)/2;
    if(a[m]==t) return m;
    else if(a[m]<t) l=m+1; else r=m-1; }
  return -1;
}
```

Time Complexities

Array: Insert O(1), Search O(n)

HashMap: Insert/Search O(1) avg

BST: O(log n), Heap: Insert/Delete O(log n)