COMP 550 · Midterm 1 Cheat Sheet

Chapters 1–3: arrays, graphs, greedy · Tue Sep 22, 65 min, closed book, 5 × 5 pts · Problem 1 = five 1-pt traces, Problems 2–5 = problem-set style (≥2 verbatim)

Rules and answer format

Indices start at 1; `A[i:j]` = A[i]..A[j]. **No hashing**: use a 0/1 array `B=[0]*n`. Graphs are adjacency lists, n vertices, m edges. Callable by name: Max-in-Array, Two-Sum, Binary-Search, Selection-Sort, Merge-Sort, BFS, DFS, Explore, Find-Cycle, Topological-Sort, Prim, Kruskal, Reverse-Delete, Select-Intervals, Fractional-Knapsack.

> **Every design answer has three labelled parts:** (1) **Algorithm**: a sentence + short pseudocode calling known algorithms. (2) **Correctness**: invariant / "returned valid, and exists returned" / case analysis / exchange argument. (3) **Running time**: iterations × cost per iteration + sort/BFS calls. Graphs: "each list scanned once O(m+n)."

Big-O: drop constants and lower terms. log n < √n < n < n log n < n^2. Connected graph: m ≥ n−1, so O(m+n)=O(m), log m = O(log n). Target O(n log n) sort then one pass. Target O(n) on sorted input two pointers. O(n log m) sort the small array, binary search each of the big.

Arrays

Max-in-Array O(n). Invariant: after iteration i, m = max(A[1:i]).

```
Two-Sum(A, t):            # A sorted, distinct
  i, j = 1, n
  while i < j:
    if A[i]+A[j] = t: return (i, j)
    if A[i]+A[j] < t: i += 1  else: j -= 1
```

O(n): each step moves a pointer; start n−1 apart. Correct: returned pair sums to t; if a solution exists it stays in A[i:j], since when the sum is too big, j pairs with nothing ≥ i (sorted), so j is safe to drop; symmetric for i. *Practice:* A=[1,3,4,7], t=10 → (2,4).

```
Binary-Search(A, t):  i, j = 1, n
  while i <= j:           # note <=
    m = (i+j)/2 rounded down
    if A[m]=t: return m
    if A[m]<t: i = m+1  else: j = m-1
```

O(log n): range halves. **Selection-Sort** O(n^2): round i swaps min of A[i:n] into i. **Merge-Sort** O(n log n): split, recurse, merge with two pointers; T(n)=2T(n/2)+O(n).

Container (PS 1.4, practice P2): maximize (j−i) · min(A[i],A[j]). i=1, j=n; record area; move the pointer at the *shorter* bar. Correct: if A[i] ≤ A[j], any (i, j′<j) is narrower and no taller, so i's best is seen. O(n).

PS1: 1.1 repeated element: sum(A) − n(n+1)/2, or seen-array. 1.2 sorted squares: pointers at both ends, fill output from the back. 1.3 intersection: sort A (m log m), binary-search each b B: O(n log m). 1.5 min difference: sort, min over adjacent pairs, O(n log n).

BFS

```
BFS(G, s):  d = [∞]*n; d[s] = 0; Q = queue(s)
  while Q not empty:
    u = dequeue
    for v in G[u]:
      if d[v] = ∞: d[v] = d[u]+1; p[v] = u; enqueue v
  return d
```

d[v] = #edges on shortest s→v path. Trace: scan lists in order, smaller label first. **O(m+n)**: each vertex dequeued once, its list scanned once; lists total m. Path to t: follow p from t back to s, reverse (≤ n steps). Adjacency matrix: edge test O(1), neighbors O(n); list: both O(out-deg u).

PS2 (BFS): 2.1 diameter: BFS from every vertex, max of max(d), O(mn). **2.3 component labels (practice P3):** `k=1; for u: if c[u]=0: BFS(G,u) labeling reached with k; k+=1.` One BFS per component, O(a+b) each O(m+n). 2.4 bipartite: BFS from 1 with c[1]=0; new neighbor gets 1−c[u]; a neighbor with c[v]=c[u] not bipartite. Colors are forced, so a clash means no coloring exists.

DFS, edge types, cycles, topological order

```
Explore(G,u): pre[u]=t; t+=1
  for v in G[u]: if pre[v]=∞: Explore(G,v)
  post[u]=t; t+=1
DFS(G): t=1; for u=1..n: if pre[u]=∞: Explore(G,u)
```

O(m+n). One clock; pre on entry, post on exit. Example G=[[2,3],[3,4],[],[1,3]]: pre=[1,2,3,5], post=[8,7,4,6].

Edge (u,v)	Timestamps
tree / forward	pre[u] < pre[v] < post[v] < post[u] (tree if Explore(v) called from u)
back (v ancestor)	pre[v] < pre[u] < post[u] < post[v]
cross	pre[v] < post[v] < pre[u] (v finished before u started)

Write directed edges as (tail, head): a back edge 2→1 is (2,1).

Find-Cycle: cycle ⇔ back edge. Return tree path v u + (u,v). Why: first-visited vertex of any cycle gets the cycle edge into it as a back edge. O(m+n).

Topological-Sort (DAG): DFS, list vertices by **decreasing post** (prepend at post time). Proof: for edge (u,v), post[u] > post[v]: if u visited first, u waits for v to finish; if v first, no v u path (DAG) so v finishes before u starts. Increasing pre fails: 1→2, 3→2 gives 1,2,3.

> **DAG pattern:** T = topo order; check each consecutive pair (T[i],T[i+1]) E by scanning G[T[i]] (each list once O(m+n)). All pairs are edges ⇔ unique topo order ⇔ Hamiltonian path = T (practice P4, PS 2.5). Some pair missing those two are mutually unreachable (PS 2.2) and swapping them gives a second order.

SCC (low priority): DFS on reversed graph; process vertices by decreasing post; BFS in G from each unlabeled vertex with a new label. O(m+n).

Exchange argument (all of Ch 3)

(1) Assume OPT ≠ ALG. (2) At the first difference, build OPT′ by swapping ALG's choice into OPT and OPT's choice out. (3) Show OPT′ feasible and *strictly better* (contradiction) or *as good and closer to ALG* (repeat until OPT′ = ALG).

Max k-Sum: sort decreasing, take k. OPT omits some top-k A[i], includes A[j] < A[i]; swap sum up by A[i]−A[j].

Waiting time (cashier): shortest job first. If OPT has longer i right before shorter j, swap: j waits A[i] less, i waits A[j] more, others unchanged; net A[j]−A[i] < 0.

Weighted completion (practice P5): minimize Σ w[i] · C_i. Sort by **p[i]/w[i] increasing**. Adjacent swap of i before j with p[i]/w[i] > p[j]/w[j]: change = w[i]p[j] − w[j]p[i] < 0. Example [(3,4),(5,1),(3,2)] → order (5,1),(3,2),(3,4), value 35.

Minimum spanning trees

Connected, undirected, distinct weights unique MST. Spanning tree = n−1 edges, connected, acyclic (any two third). Cut = vertex set S; edge crosses if one endpoint in S.

Cut property: lightest edge crossing any cut is in the MST. *Proof:* if not, T+e has one cycle; it re-crosses the cut at some f T with w(f) > w(e); T+e−f is a lighter spanning tree. (Must pick f *on the cycle*, else T+e−f may disconnect.)
Cycle property: heaviest edge on any cycle is not in the MST. *Proof:* if f T, T−f gives a cut; the cycle re-crosses at some e with w(e) < w(f); T−f+e is lighter. (Must pick e *crossing that cut*, else T−f+e may have a cycle.)

Algorithm	Rule	Why	Time
Prim	from S={1}, add lightest edge leaving S, n−1 times	cut	O(mn)
Kruskal	edges lightest first; keep unless it closes a cycle (BFS on forest F: O(n))	cut	O(m log n + mn) = O(mn)
Reverse-Delete	edges heaviest first; delete unless G−e disconnects (BFS O(m))	cycle	O(m²)

Kruskal correctness: when e={u,v} is added, S = u's component in F; e is the lightest edge crossing S. Reverse-Delete: a removable edge is the heaviest on a cycle; nothing acyclic remains uncut.

Intervals and knapsack

```
Select-Intervals(A): sort by end time; S = []
  for e in A: if e starts after last(S) ends: add e to S
```

O(n log n) (compare only with the last chosen). Proof: first index i where ALG[i] ≠ OPT[i]: OPT[i] ends no earlier than ALG[i] (ALG picked the earliest-ending compatible one), so replace OPT[i] by ALG[i]; still feasible, same size; repeat. Wrong rules: shortest ([1,10],[9,12],[11,20]), earliest start, fewest conflicts (11-interval example). Endpoints count as conflicts.

```
Fractional-Knapsack(v,w,B): sort by v[i]/w[i] decreasing
  for each i: x[i] = min(1, remaining/w[i])
```

O(n log n); ≤ 1 fractional item. Proof: first i with OPT[i] < ALG[i]; OPT uses some later j; move ε weight from j to i (x[i] += ε, x[j] −= ε · w[i]/w[j]); weight unchanged, value changes by ε · w[i] · (v[i]/w[i] − v[j]/w[j]) ≥ 0; repeat. Example v=[4,1,2], w=[5,1,4], B=7 → x=[1,1,¼], 5.5. **Fails for 0/1:** v=[4,4,7], w=[5,5,6], B=10: greedy 7, best 8.

Running times

Max-in-Array, Two-Sum	O(n)	BFS, DFS, Find-Cycle, Topo-Sort, SCC	O(m+n)
Binary-Search	O(log n)	Prim, Kruskal	O(mn)
Selection-Sort	O(n²)	Reverse-Delete	O(m²)
Merge-Sort, Max-k-Sum	O(n log n)	Select-Intervals, Frac-Knapsack	O(n log n)

Practice midterm answers

Graph G=[[4],[1,3,4],[5],[5],[2]]. 1a (2,4). 1b d=[0,3,4,1,2]. 1c post=[10,7,6,9,8] (pre=[1,4,5,2,3]; tree path 1→4→5→2→3). 1d back: (2,1),(2,4),(3,5). 1e cross: none. P2 container. P3 component labels. P4 topo sort + consecutive edges. P5 sort by p/w + adjacent swap.