# CS2106 Tutorial 7

AY 25/26 Sem 1 — github/omgeta

Q1. Code:

```
function allocate(X) {
  n = ceil(X / UNIT_KB)
  for (i from 0 to bitmap.size - 1)
    for (j from i to bitmap.size - 1)
      if (j - i + 1 == n) return addr(bitmap[i])
  return FAIL
}

function free(Y, X) {
  n = ceil(X / UNIT_KB)
  start = Y / UNIT_KB
  for (i from start to start+n-1)
    bitmap[i] = 0
}

// merge not needed
```

Q2. First Fit: 400KB (left 33KB) → 600KB (left 390KB) → 500KB (left 32KB) → FAIL
Best Fit: 400KB (left 33KB) → 250KB (left 40KB) → 500KB (left 32KB) → 600KB (left 109KB)
Worst Fit: 600KB (left 243KB) → 500KB (left 210KB) → FAIL → FAIL

Best Fit is the most memory efficient with smallest internal fragmentation, First Fit is the fastest, and Worst Fit could work with merging

Q3. Can reduce runtime by skipping early areas which were not free. However, memory usage is similar in its potential for large internal fragmentation, and even worse considering its tendency to skip smaller holes before the start point.

Q4. `0: A[256], 4: Free[256], 8: Free[512]` (Allocate A)
`0: A[256], 4: B[64], 5: Free[64], 6: Free[128], 8: Free[512]` (Allocate B)
`0: A[256], 4: B[64], 5: Free[64], 6: C[128], 8: Free[512]` (Allocate C)
`0: A[256], 4: B[64], 5: Free[64], 6: C[128], 8: D[128], 10: Free[128], 12: Free[256]` (Allocate C)
`0: Free[256], 4: B[64], 5: Free[64], 6: C[128], 8: D[128], 10: Free[128], 12: Free[256]` (Free A)
`0: Free[256], 4: B[64], 5: Free[64], 6: Free[128], 8: D[128], 10: Free[128], 12: Free[256]` (Free C)
`0: Free[512], 8: D[128], 10: Free[128], 12: Free[256]` (Free A)

Q5. (a.) Number of partitions = 16MB / 4KB = 4096 so maximum and minimum overhead with bitmap is 4096 bits = 512 B

(b.) Each node is start (4 B), size (4 B), status (1 B) and next (4 B) for 13 B.
Minimum with a single node is 13B. Maximum when each 1KB is a partition is 16 MB /1 KB $\times$ 13 B = 212992 B $\approx$ 208 KB

(c.) Number of allocatable units = 16MB / 1KB = 16384 bits so maximum and minimum overhead with bitmap is 16384 bits = 2 KB