Question 1: Your Backpack

1. (a) I-cache spatial locality > D-cache spatial locality.
   The data-cache has no spatial locality since they are random accessed.
   Instruction accesses are highly sequential, since there are very few branches, and
   even the branches contribute to spatial locality since they branch no more than
   5 instructions back.

   (b) I-cache spatial locality > I-cache temporal locality.
   I-cache temporal locality is only when there is a branch. Since there are very few,
   this is lower than the spatial locality.

   (c) I-cache hit rate > I-cache miss rate.
   Hit rate = 1 - miss rate. Since I-cache spatial locality is extremely high, the hit
   rate is likely to be very high as well. Assuming a cache line can hold more than 1
   instruction! If a cache line only fits one instruction, spatial locality doesn't help.

   (d) I-cache hit rate > D-cache hit rate.
   The data cache has a terrible hit rate – loads are almost always misses because
   they are random. Stores might always be hits, but if the loads evict data from
   other loads, stores might miss too.

   (e) D-cache miss rate > D-cache hit rate.
   Same reason as above.

   (f) CPI for stores > CPI for loads.
   The loads are almost always guaranteed of missing in the cache. The stores might
   hit, depending on how close they are to the load from the same address.

2. Size of I-cache or D-cache:
   Size of cache line: $2^4 = 16$ bytes.
   Number of cache lines: $2^{11} = 2048$
   Total size: $16 \times 2048 = 32KB$
   Tag space: $17 \times 2048 = 34Kbits$

3. (a) Increasing the offset and decreasing the index keeps the cache size the same.
    There are fewer cache lines, but they are longer. Longer cache lines expand
spatial locality (by holding values from more adjacent addresses). Fewer cache lines can lead to more conflict misses, so the window for temporal locality is worse (imagine having only 1 cache line – even if a program has high temporal locality, if there is one other address in between two of the same addresses, you get no locality from the cache).

For this machine, the I-cache accesses (instruction addresses) have high spatial locality, so this would increase the hit rate. The D-cache accesses have bad temporal locality, so with fewer cache lines, there would be more conflict misses and therefore decrease the hit rate.

(b) With no offset bits, the cache lines hold 1 byte. The number of cache lines is the same, so the cache size is reduced. This would decrease the hit rate for both caches, because of the drastic reduction in cache size. It would hurt the instruction-cache more, however, because its spatial locality is useless.

**Question 2: Too Small a Backpack**

- Offset: 4 bits (for 16B lines)
- Index: 8 bits (for 256 lines)
- Tag: 20 bits
- Access Pattern:
  - 0x1000: Miss (index=0)
  - 0x1004: Hit (same cache line)
  - 0x1010: Miss (index=1)
  - 0x11c0: Miss
  - 0x2000: Miss (index=0, but second way)
  - 0x21c0: Miss (second way)
  - 0x3400: Miss
  - 0x3404: Hit
  - 0x3f00: Miss
  - 0x2004: Hit
  - 0x1004: Hit

1. There are 7 cache misses, all compulsory. No evictions.

2. Since there are no evictions, the eviction policy doesn’t affect the number of cache misses.

3. If the cache was direct-mapped, you would have double the cache lines, so 1 extra index bit. This would cause 0x1000 and 0x2000 to still go to different cache lines, so it would not affect the miss-rate.

4. There was some information missing here. Assume writing data to L2 takes 10 cycles and reading data from L2 takes 10 cycles.
Given that, on a read miss, you need to read data from L2: 10 cycles. You need to write that somewhere in the cache. Assuming that line was dirty, you have to write that data back first: 10 cycles. You could potentially do both at the same time though.

For a write miss, you write the old data back first (10 cycles), then write data to the cache line, and it stays dirty.

5. The read miss works the same way, except the cache line being evicted is clean, so it doesn’t need to be written to L2.

On a write miss, you are still evicting a clean cache line, so it doesn’t need to be written to L2. However, since it is write-through, the new data needs to be written to L2 (10 cycles). Also, since it is write-allocate, the new data needs to be written to L1 as well (which is fast, so it doesn’t matter in terms of time - it does however cause an eviction, and potentially future conflict misses).

**Question 3: Los Angeles**

1. Fraction serviced from L2 = (Miss in L1)*(Hit in L2) = 0.5*0.7 = 35%
   Fraction serviced from memory = (L1 miss)*(L2 miss)*(Hit in memory) = 0.5*0.3 = 15%

2. Miss rate for L2 = 1 - Hit rate = 30%
   Miss time = Hit in main memory = 200 cycles.

3. Miss rate for L1 = 1 - Hit rate = 50%
   Miss time = (L2 hit rate)*(L2 hit time) + (L2 miss rate)*(L2 miss time)
   Miss time = 0.7*15 + 0.3*200 = 70.5 cycles.

4. New main memory hit time = 200/1.1 = 182 cycles.
   New L1 miss time = 0.7*15 + 0.3*182 = 65.1 cycles.
   Improvement = 70.5/65.1 = 8.3%

5. L1 miss time = 200 (going to main memory). Note that this is much worse.
   However, L1 average service time = (hit rate)*(hit time) + (miss rate)*(miss time)
   Service time = 0.75*2 + 0.25*200 = 51.5 cycles, which is an improvement.

**Question 4: Cache Design**

1. Dictator: 1024 1B lines = 1KB.
   Oligarch: 128 4B lines x 2 ways = 1KB
   Democle: 256 4B lines = 1KB
2. Dictator: 1024 x 6-bit tags = 6Kbit
   Oligarch: 256 x 7-bit tags = 1792 bit
   Democle: 256 x 14-bit tags = 3584 bit

3. Dictator probably has the most conflict misses, since it is direct mapped. Democle, because it is fully associative, can never have conflict misses.

4. Since the caches are the same size and the reason in the previous answer, Dictator is 50%, Oligarch is 70%, and Democle is 90%.

5. Based on the previous answer, average service time:
   Dictator: 0.5*1 + 0.5*20 = 10.5 cycles
   Oligarch: 0.7*2 + 0.3*20 = 7.4 cycles
   Democle: 0.9*5 + 0.1*20 = 6.5 cycles.