Sample Programs:

```
04cr.c
for (unsigned int c = 0; c < SIZE; c++) {
    for (unsigned int r = 0; r < SIZE; r++) {
        array[(r * SIZE) + c] = (r * SIZE) + c;
    }
}
```

```
04rc.c
for (unsigned int r = 0; r < SIZE; r++) {
    for (unsigned int c = 0; c < SIZE; c++) {
        array[(r * SIZE) + c] = (r * SIZE) + c;
    }
}
```

Running Times:  

- **04cr.c** (Program #1):
- **04rc.c** (Program #2):

Caching Strategies: Keeping Data Close

In working with memory in any computer system, we want to access it as quickly as possible. However, space is extremely limited in the fastest memory, so we need strategies on what data to keep close. General Purpose Memory:

- CPU Registers:
- CPU Cache (i7-12700K, Released Q4’21):
- RAM:

Key Idea: Locality of Reference

System Memory: Limited, Shared, and Simple

1. 
2. 
3. 

To help us to begin to organize this RAM, we divide the RAM up into chunks called ________.

On Linux, find the size of a page:
```
# getconf PAGESIZE
- On almost every modern system, a page is _____ KB.
```

Virtual Memory:

Modern systems provide an abstraction between the ________ and ________:

1. A ___________ translates a ___________ into a physical address.
2. Every memory address is made up of the ___________ and the ___________:
3. Virtual Memory is **NOT shared** between processes/apps:
4. **EVERY memory address** you have ever seen is a virtual memory address!
Let’s explore a sequence of allocations using a page table:

<table>
<thead>
<tr>
<th>P1 Page Table:</th>
<th>RAM</th>
<th>P2 Page Table:</th>
<th>P3 Page Table:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Allocation Sequence:
1. Process #1 (P1): a = malloc(3 * 4096)
2. Process #3 (P3): b = malloc(5 * 4096)
3. Process #1 (P1): c = malloc(2 * 4096)
4. Process #3 (P3) exits.
5. Process #2 (P2): d = malloc(4 * 4096)
6. Process #2 (P2): e = malloc(5 * 4096)
7. Process #1 (P1): a = realloc(a, 5 * 4096)

Advantages of a Virtual Memory System:
1.

```
05.c
16  printf("  Start of 'array': %p\n", array);
17  printf("    End of 'array': %p\n", &(array[(SIZE * SIZE) -1]));
```

2.

3.

---

Simple Simulation of Page Tables with Disk Pages

<table>
<thead>
<tr>
<th>RAM:</th>
<th>P1 Page Table:</th>
<th>Disk Pages:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]:</td>
<td>[0]:</td>
<td>/programCode (1/5)</td>
</tr>
<tr>
<td>[1]:</td>
<td>[1]:</td>
<td>/programCode (2/5)</td>
</tr>
<tr>
<td>[2]:</td>
<td>[2]:</td>
<td>/programCode (3/5)</td>
</tr>
<tr>
<td>[3]:</td>
<td>[3]:</td>
<td>/programCode (4/5)</td>
</tr>
<tr>
<td>[4]:</td>
<td>[4]:</td>
<td>/programCode (5/5)</td>
</tr>
<tr>
<td>[5]:</td>
<td>[5]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[6]:</td>
<td>[6]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[7]:</td>
<td>[7]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[8]:</td>
<td>[8]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[9]:</td>
<td>[9]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[10]:</td>
<td>[10]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[11]:</td>
<td>[11]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[12]:</td>
<td>[12]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[13]:</td>
<td>[13]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[14]:</td>
<td>[14]:</td>
<td>/programImage.png</td>
</tr>
<tr>
<td>[15]:</td>
<td>[15]:</td>
<td>/programImage.png</td>
</tr>
</tbody>
</table>

1: Load Program
2: Run PC, pg1:
   - malloc(4000)
3: Run PC, pg2:
   - malloc(10000)
   - Open hiddenImage.png
   - Read all of image
4: Run PC, pg3:
   - Access OG 4 KB
   - Finish program

Q1: What is the range of possible file sizes for hiddenImage.png?

Q2: What is the range of possible file sizes for ./programCode?

Q3: What is the size of the heap immediately before the program finishes?