1.

Þetta tæki **64** síður. Það er vegna þess að

- hvert `int` er **4** byte
- síða getur haldið **1024** bætum
- það eru **256** `int` á hverri síðu
- hver lína í fylkinu hefur **128** heiltölur
- $\frac{256}{128} = 2$
- $\frac{128}{2} = 64$

ergo: við þurfum **64** síður :)

2.

Ef rammar eru tómir í upphafi þá erum við að minnsta kosti með 64 skelli og þar sem við förum yfir eitt stak í hverjum ramma áður en við förum í næsta stak, en vegna þess að við tökum í raun bara tvö stök fyrir hvern ramma þá erum við að page faulta þessa 64 ramma 128 sinnum, þe við fáum $128 * 64$ pagefault

3.

hér höfum við upprunalega kóðann, ekki optimized >:(

```
int i, j;
int data[128][128];
for (j = 0; j < 128; j++) {
  for (i = 0; i < 128; i++) {
    data[i][j] = 0;
  }
}
```

hér eru tvær mismunandi lausnir á þessu vandamáli

3. 1.

```
int i, j;
int data[128][128];
for (i = 0; i < 128; i++) {
  for (j = 0; j < 128; j++) {
    data[i][j] = 0;
  }
}
```

3. 2.

```
int i, j;
int data[128][128];
for (j = 0; j < 128; j++) {
  for (i = 0; i < 128; i++) {
    data[j][i] = 0;
  }
}
```

4.

Ef við notum nýja kóðann þá sjáum við að við klárum allar breytur í hverjum ramma áður en við förum í næsta þetta minnkar því pagefault hjá okkur um **128** þ.e. við endum með **64** pagefault