Exercises from old exams

Memory Management 3 (exercises on ch. 10)

1. Q. Let's consider the following string of page references: 577517111434123.

   Simulate the behaviour of a working-set page replacement strategy (exact version) with time window Δ = 3. Compute how many page faults (accesses to pages not in the resident set) are generated by the string, and by which references. Please represent the resident set after each reference.
   We want to measure the locality of the program (that generated the reference string), based on the so called "Reuse Distance". The Reuse Distance at time T_i (RD_i), when page P_i is referred, is defined as the number of (distinct and different from P_i) pages in the reference string during the interval from the previous reference to P_i to T_i. Let's conventionally assume that, for the first access to a given page, we count all previously accessed (distinct) pages. For instance, at time 3, second access to page 5, RD_3 = 1, as a single page (7) was accessed to between the two accesses to page 5. Given the RD_i values for all rimes, compute their average RD_{avg}. Finally, let's define a locality ratio for the program, given by L = 1 / (1+ RD_{avg}). Compute all the RD_i values, then RD_{avg} and L.

   A.

References	5	7	7	5	1	7	1	1	1	4	3	4	1	2	3
Resident Set	5	5	5	5	5	5				4	4	4	4	4	3
		7	7	7	7	7	7	7			3	3	3	2	2
					1	1	1	1	1	1	1		1	1	1
Page Fault	x	x			x					x	x		x	x	X
Page Out							x		x			x		x	X
RD	0	1	0	1	2	2	1	0	0	3	4	1	2	5	3

Total number of page faults: 8….. of page outs: 5… RD_{avg}: 25/15=5/3=1,67 L: 1/(1+5/3)=3/8=0,375

2. Let's consider the following string of page references: *3, 4, 1, (3, 1, 4, 4, 3, 1, 1)*10,* where the syntax (…)*n is used to denote n repeated instances of the string within round brackets (the string could be the result of an iterative construct).

   Simulate the behaviour of a working-set page replacement strategy (exact version) with time window Δ = 3. Compute how many page faults (accesses to pages not in the resident set) are generated by the string, and by which references. Please represent the resident set after each reference, up to the first two iterations of the sub-string in round brackets (the sub-string is actually repeated 10 times, so just explicitly represent the 2 initial iterations).
   WARNING: each resident set row represents a distinct frame, so a page that resides in a frame shouldn't change row in two adjacent columns. Whenever a page-fault occurs with no page-out (a new frame ais simply added to the resident set) Choose the firs free frame in descending (top to bottom) order. When a page-fault and a page-out occur simultaneously, the page-out frame os re-used for the page-faulting page. When a page-out and a page-fault simultaneously occur on the same page, the replacement algorithm should be clever enough to keep the resident set as it is.

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
References	3	4	1	3	1	4	4	3	1	1	3	1	4	4	3	1	1
Resident Set	3	3	3	3	3	3			1	1	1	1	1	1	3	3	3
		4	4	4		4	4	4	4				4	4	4	4	
			1	1	1	1	1	3	3	3	3	3	3			1	1
Page Fault	x	x	x			x		x	x				x		x	x	
Page Out					x		x	x		x				x	x		x

   The resident set in the 8 additional iterations will be
   - Equal to the 10-16 interval (repeated 8 times)? (YES/NO motivate)

No. Although the period is 7, pages 1 and 3 are not in the same frame at the start and end times of a period

- or it will repeat 4 times the 3-16 interval? (YES/NO motivate)
  No. Though the interval is a union of the two initial iterations, the first time frame has a different behaviour: during the first iteration, page 4 is in the resident set, wheras it is out at the initial time of all successive iterations.
- or it will show another configuration (which one?)?
  The 3-16 interval is repeated again 4 times, provided that we remove page 4 and the page-out from the first time. Or (let's say it again in another way) the 10-16 interval is repeated 8 times, by alternating the contents of the first and the memory frame, at each iteration (odd/even).

How many page faults and page-outs will be triggered by the reference string (including non represented iterations, that is the missing 8 iterations)?
9 PF and 7 PO in the visualized part, 8*(3 PF and 3 PO) in the missing part: overall 33 PF and 31 PO

3. *Q* Consider the following program fragment, performing a matrix-based computation*:*

```
...
float M[512][512],V[512];
...
for (i=0; i<512; i++) {
 V[i]=0;
 for (j=0; j<=i; j++) {
   if (i%2==0) {
     V[i] += M[i][j];
   }
   else {
     V[i] -= M[i][511-j];
   }
 }
}
...
```

an array of 512 rows. For each iteration on i we initialize v of i
One entry in the v array. And we do a loop for j starting 0 ending i.
for the odd and even values of i we either visit a row starting at position 0 or starting at the last postition 511. When i = 0 we will use.
this is the visit of the triangular part of the matrix, and this would be visiting the hight part of the traingular matrix. So it essentially means if the

The machine code generated from the program is executed on a system with memory management based on demand paging, 2Kbyte pages, page replacement driven by a SECOND CHANCE policy.
Let's assume that:

- the size of a `float` is 32 bits
- the code segment (machine instructions) has size less that one page
- `M` and `V` are allocated at contiguous logical addresses (`M` first, then `V`), starting at logical address 0x5524AE00
- the `M` matrix is allocated following the "row major" strategy, that is by rows (first row, followed by second row, …).

How many pages (and frames) are needed to store the matrix and the array?

|V| = 512*sizeof(float) = 2KB = 2KB/2KB pages = 1 page
|M| = 512*512*sizeof(float) = 1MB = 1MB/2KB pages = 0.5K pages = 512 pages
The starting address 0x5524AE00 is NOT a multiple of the page size (it should end with eleven 0 bits), it rather starts at ¾ of a page. We thus need to slightly correct/modify our previous results: V overlaps 2 pages and M 513 pages (the first one shared with V): 514 overall.

- Suppose now that variable i and j are allocated in registers (accessing them does not produce memory references), how many memory references (for reading and writing data) produces the proposed program (do not consider instruction fetches)?

Notation: N_i number of iterations of the outer for
N_j number of iterations of the inner for
Solution

```
for (i=0;i<512;i++){ // Ni = 512 iterations
V[i]=0;              // 1 Write per iteration, total: Ni(512) writes
for (j=0;j<=i;j++){  // Nj = i+1 iterations, repeated Ni(512) times (i changes)
V[i] += M[i][j];     // Nj = Σi=0..511(i+1) = 512*(512+1)/2 = 128K+256 = 131328
```

```
    Or                // for each iteration: 2 Reads and 1 Write
V[i] += M[i][511-j]; // 2*Nj reads, Nj writes
```

WARNING: The computations showed above are enough. For sake of completeness, we show below a detailed computation for odd/even iterations.

```
V[i] += M[i][j];       // iterations with even i: Nj,0 = 131328/2 = 65664 (*)
                       // for each iteration 2 Reads, 1 Write
                       // overall 2*Nj,0 read, Nj,0 write
V[i] += M[i][511-j];   // iterations with odd i: Nj,1 = 131328/2 65664 (*)
                       // for each iteration 2 Reads, 1 Write
                       // overall 2*Nj,1 reads, Nj,1 writes
```

(*) the result is approximated. We are assuming that the number of iterations on j with even i and the one with odd I are equal. The exact computation, (assuming `k=i/2`) is:

$N_{j,0} = \Sigma_{k=0..255}(2k+1) = 1+3+5\ldots+511 = 2*\Sigma_{k=0..255}(k+1)-256 = 256*(256+1)-256 = 256^2 = 65536$

$N_{j,1} = \Sigma_{k=0..255}(2k+2) = 2+4+6\ldots+512 = 2*\Sigma_{k=0..255}(k+1) = 256*(256+1) = 256*257 = 65792$

Alternative solution. An alternative and (perhaps) simpler approach follows. The total number of iterations N_j is equal to the size of a lower triangular matrix (diagonal included). Actually, iterations with even `i` value visit a row by increasing column indexes, whereas those with odd `i` go in the opposite direction. Nevertheless, the total number of inner iterations is independent of the direction: (matrix size)/2 + (half a diagonal) = 512*512/2 + 256.

(Derailed computation for even/odd `i` values, NOT needed): the number of iterations with increasing column index is slightly smaller (for each pair of even/odd `i` values, the second one has one more inner iteration), so overall, odd `i` iterations are 256 more than the even ones). More in detail:

$2*N_{j,0}+256 = N_j = 128K+256$

$2*N_{j,0} = 128K$

$N_{j,0} = 64K$

$N_{j,1} = N_{j,0} + 256 = 64K + 256$

- Let N_T be the total amount of memory references to data (we omit instruction fetches for sake of simplicity) and N_L be the number of references to a page already accessed to within the previous 10 references. We define a measure of (data) **locality** for the program as the ratio $L = N_L/N_T$. Compute the locality of the proposed program.

WE SOLVE THE EXERCISE IN A SIMPLIFIED WAY, ASSUMING PAGE ALIGNMENT. THE EXACT SOLUTION WOULD NEED MINIMAL MODIFICATIONS

Every row in M fits exactly into one page, so visiting it left-to-tight or right-to-left makes no difference on locality and page-faults. Locality and page faults can thus be computed on the following (simplified and equivalent) code:

```
for (i=0; i<512; i++) {
   V[i]=0;
   for (j=0; j<=i; j++) {
     V[i] += M[i][j];
   }
}
```

M references (for reading)

For each `i` value (for each row of M), the first reference (j=0) is not local (it is on a new page/row, not accessed before), whereas all other accesses are local. perché accede a una nuova pagina (nuova riga), Overall, 512 non local references.

V references (for reading and writing)

V fits in one page, so the first reference is not local, whereas all other references are local. Overall, 1 non local access

Total (M+V) NON local references: 1+512 = 513

$N_T = N_i$ (`V[i] = 0 initializations`) $+ 3*N_j$ (inner iterations: 2 Reads + 1 Write)

$N_L = N_T - 513$ (NON local references)

$L = L = N_L/N_T = (N_T - 513)/N_T = (512 + 3*(128K + 256) - 513) / (512 + 3*(128K + 256)) =$

$= 1 - 513 / (512 + 3*(128K + 256)) \approx 1 - 512/3*128K = 1 - 1/3*256 =$

$= 1 - 0{,}0013 = 9{,}9987$

- Compute the number of page faults generated by the proposed program. Assume that **10 frames** have been allocated, one reserved to instructions, so 9 frames are available for data. (motivate the answer)

No detailed simulation is needed, as (due to the high locality of the program) page faults can be easily estimated.

V: 1 page fault (2 page faults if we also consider misalignment to page boundaries)
M: 1 page fault for each row/page (total: 512). Then all other page references are local and do not produce page faults.
Total page fault number: 1+512 = 513 page faults *(2 + 512 = 514, when we consider page misalignment)*