Problem #1

RESULTS:

Table 1:Inclusive Policy

File name	**L1 misses**	**L2 misses**	**L3 misses**
bzip2.log_l1misstrace	1,06,57,627	53,98,166	14,46,388
gcc.log_l1misstrace	1,46,10,811	30,36,461	13,73,402
gromacs.log_l1misstrace	34,31,511	3,36,851	1,70,531
h264ref.log_l1misstrace	23,48,573	9,69,678	3,42,146
hmmer.log_l1misstrace	35,09,765	17,43,421	3,91,226
sphinx3.log_l1misstrace	1,07,53,447	88,20,349	82,07,362

Table 2:NINE Policy

File name	**L1 misses**	**L2 misses**	**L3 misses**
bzip2.log_l1misstrace	1,06,57,627	53,97,576	14,45,846
gcc.log_l1misstrace	1,46,10,811	30,29,809	13,66,248
gromacs.log_l1misstrace	34,31,511	3,36,724	1,70,459
h264ref.log_l1misstrace	23,48,573	9,65,624	3,33,583
hmmer.log_l1misstrace	35,09,765	17,35,322	3,76,344
sphinx3.log_l1misstrace	1,07,53,447	88,15,130	82,05,144

Table 3:Exclusive Policy

File name	**L1 misses**	**L2 misses**	**L3 misses**
bzip2.log_l1misstrace	1,06,57,627	53,97,576	8,89,221
gcc.log_l1misstrace	1,46,10,811	30,29,809	12,42,824
gromacs.log_l1misstrace	34,31,511	3,36,724	1,59,302
h264ref.log_l1misstrace	23,48,573	9,65,624	1,43,681
hmmer.log_l1misstrace	35,09,765	17,35,322	3,00,046
sphinx3.log_l1misstrace	1,07,53,447	88,15,130	72,20,776

OBSERVATIONS:

1.Why we observe more misses in inclusive cache than exclusive and NINE?

As we can observe from the above tables, the misses in the inclusive policy is greater as compared to exclusive and NINE policy.The reason behind such result is that whatever blocks are accessed, they are limited by the size of L3 cache.i.e. the unique block addresses that can be accessed is determined by L3 cache size.L3 is accessed only when block is not found in L2 and in case of Inclusive policy L2 contains a part of L3 cache, therefore the chances that the block is present in L3 is less than in the case of NINE and Exclusive, where L3 might not and certainly doesn't contain any part of L2 respectively. Hence, the order of L3 misses is Inclusive cache>NINE cache>Exclusive cache.
If we compare L2 misses in inclusive cache with NINE/exclusive, it seems that L2 caches in inclusive are large because in the case of inclusive cache, whenever the block is evicted from L3 cache it is invalidated from L2 cache and hence if that evicted block's request is made,we get miss in L2.But for the case of NINE/exclusive,the block may be present in L2 cache.

2.*Normal assumption is that if block exits L3, it is less frequent,so why does keeping it in L2 make sense? And how does it then reduce L2 misses?*

The block is evicted from L3 means it is not accessed from a long time .But it might be the case that just after evicting that block,a request from that can occur.Hence keeping it in L2 helps to reduce the misses in L2, albeit by not a large number, as we can see from the table as it isn't very probable.

3.Why L2 misses are same in exclusive and NINE?

We can see that the misses in L2 in exclusive and NINE policies are the same .This is because of the fact that in NINE policy, when we get a miss in L2 cache we fill it either from L3 cache(in case of L3 hit) or from the memory(in case of L3 miss). In case of exclusive policy,we transfer the block from L3 and invalidate it in L3(in case of L3 hit) or fetch it from memory directly into the L2 cache(in case of L3 miss). So, in effect, both the policies work in the same fashion when it comes to filling up the L2 cache, if we ignore what happens with the L3 cache.

4.Why L3 misses are greater in NINE compared to exclusive policy?

If we consider the NINE policy ,when the block is replaced from L2 cache at some instant and suppose due to replacement in L3 cache that particular block is evicted from L3 also. Now if again that block is accessed, it causes misses in L2 and L3 both.
Consider the same case for exclusive policy, at the time of replacement of block from L2 cache, that block is separately stored in L3 cache.Even if there is eviction in L3 cache ,as this is the not the 'least recently used block',it will not get evicted. Now if in the subsequent requests, there is request for this block ,it will be a hit and hence it causes less misses in L3 in case of exclusive policy as compared to NINE policy.
Eg. Suppose there are 2 blocks in L2 cache and 4 blocks in L3 cache.
Following block accesses are carried out in case of NINE cache:

1.Fill X
2.Fill Y
3.Fill M → causes replacement of X in L2
4.Fill N → causes replacement of Y in L2
5.Fill P → It causes replacement of M in L2 and X in L3.
6.Fill Q → It causes replacement of N in L2 and Y in L3(L3 is full).
7.Request X → It causes miss in L2 and L3

Consider the same scenario for exclusive cache:
1.Fill X
2.Fill Y
3.Fill M → causes replacement of X in L2,places X in L3
4.Fill N → causes replacement of Y in L2 ,places Y in L3
5.Fill P → It causes replacement of M in L2 and places M in L3.
6.Fill Q → It causes replacement of N in L2 and places N in L3(L3 is full).
7.Request X → It causes miss in L2 and but hit in L3 .

Problem #2

RESULTS:

Table 4:Classification of L3 misses (LRU policy)

File name	**Cold misses**	**Capacity misses**	**Conflict misses**
bzip2.log_l1misstrace	1,19,753	12,41,648	84,987
gcc.log_l1misstrace	7,73,053	5,96,871	3,478
gromacs.log_l1misstrace	1,07,962	61,406	1,163
h264ref.log_l1misstrace	63,703	2,72,177	6,266
hmmer.log_l1misstrace	75,884	3,01,140	14,202
sphinx3.log_l1misstrace	1,22,069	82,65,179	-1,79,886*

Table 5:Classification of L3 misses (Belady's optimal policy)

File name	**Cold misses**	**Capacity misses**	**Conflict misses**
bzip2.log_l1misstrace	1,19,753	4,17,084	9,09,551
gcc.log_l1misstrace	7,73,054	1,66,236	4,34,112
gromacs.log_l1misstrace	1,07,962	35,292	27,277
h264ref.log_l1misstrace	63,703	47,903	2,30,540
hmmer.log_l1misstrace	75,885	77,563	2,37,778
sphinx3.log_l1misstrace	1,22,069	29,46,511	5,138,782

OBSERVATIONS:

1.Can LRU replacement outperform for set associative as compared to fully associative?

As observed from Table 4, negative value of L3 conflict misses for *sphinx3.log_l1misstrace**, we can say that misses in set associative cache are less than fully

associative cache. There are some patterns for LRU replacement policy in which set associative cache can outperform fully associative cache.
For example, we can consider block access : 1,3,5,7,2,4,6,8,7,5,3,1,8,6,4,2 for fully associative cache with 4 blocks and 2 way set associative cache with total 4 blocks.

Line 0	~~1~~ ~~2~~ ~~7~~ 8
Line 1	~~3~~ ~~4~~ ~~5~~ 6
Line 2	~~5~~ 6 ~~3~~ 4
Line 3	~~7~~ ~~8~~ ~~1~~ 2

Fully associative

cold misses:8
capacity misses:8

Set 0	~~2~~ ~~6~~ 2
	~~4~~ ~~8~~ 4
Set 1	~~1~~ ~~5~~ 1
	~~3~~ ~~7~~ 3

2-way set associative

cold misses=8
capacity misses=4
conflict misses=0

Here actually the conflict misses are zero,but if we apply the zero conflict definition,
Conflict misses = (Total misses in set associative cache) – (Total misses in fully associative cache)

= 12-16
= -4

2.Why capacity misses are lesser in Belady's optimal policy as compared to LRU policy?

Belady's optimal policy replaces the block which is not going to be used for longer period of time in future. There are high chances that we get lots of hits for the other blocks in between unlike LRU which replaces the block that was used longest time ago but it might so happen that just after its eviction, that particular block is requested again. Hence capacity misses are lesser for Belady's optimal policy as compared to LRU policy.

And since capacity misses are lesser in Belady's optimal policy than in LRU policy, therefore, the next column which shows that conflict misses in Belady's is more than that of LRU's makes sense as well. This is because of the fact that cold misses in both policies are defined in the same way and hence are equal and conflict misses are calculated as (X-capacity misses) where X is the total number of misses in L3 set associative cache.