Task 1

	code	data	within	out
simpleloop	64	2574	172651	210339
matmul	74	1017	62973520	211728
blocked	77	1018	67664111	212193
make sim	136	132		
gcc	111	120		

The table shows that most of the programs access the data more than the code and the numbers are close however total references are different. Program access so many addresses outside of the markers.

Task 2

simpleloop

		50	100	150	200
fifo	Hit rate:	99.2227	99.2820	99.2924	99.2945
	Miss rate:	0.7773	0.7180	0.7076	0.7055
lru	Hit rate:	99.2749	99.3008	99.3013	99.3013
	Miss rate:	0.7251	0.6992	0.6987	0.6987
clock	Hit rate:	99.2754	99.3010	99.3015	99.3015
	Miss rate:	0.7246	0.6990	0.6985	0.6985
opt	Hit rate:	99.3036	99.3125	99.3125	99.3125
	Miss rate:	0.6964	0.6875	0.6875	0.6875

matmul-100

		50	100	150	200
fifo	Hit rate:	98.2121	98.2845	99.9455	99.9464
	Miss rate:	1.7879	1.7155	0.0545	0.0536
lru	Hit rate:	98.3521	98.4072	99.9480	99.9480
	Miss rate:	1.6479	1.5928	0.0520	0.0520

Hit rate:	99.9821	99.9830	99.9734	99.9737
clock				
Miss rate:	0.0179	0.0170	0.0266	0.0263
Hit rate:	99.0703	99.8531	99.9579	99.9695
opt				
Miss rate:	0.9297	0.1469	0.0421	0.0305

		blocked-100-25		
	50	100	150	200
Hit rate:	99.9905	99.9936	99.9938	99.9953
fifo				
Miss rate:	0.0095	0.0064	0.0062	0.0047
Hit rate:	99.9924	99.9944	99.9944	99.9946
lru				
Miss rate:	0.0076	0.0056	0.0056	0.0054
Hit rate:	99.9929	99.9941	99.9943	99.9953
clock				
Miss rate:	0.0071	0.0059	0.0057	0.0047
Hit rate:	99.9945	99.9956	99.9963	99.9966
opt				
Miss rate:	0.0055	0.0044	0.0037	0.0034

The implementation of the FIFO is the easiest one, because I use the circular-queue(FIFO) to keep track of address. The program delete from the front of the queue and add the end of the queue. But FIFO algorithm has the biggest miss rate, because what it basically does is take the oldest element and evict it. But since programs might access the same address frequently, so faults increase. In implementation of LRU i use the array that stores the frame numbers and sort it whenever new address is added and hit happens. The first index of array is always contains the least recently used address. When miss and hit happens the sorting algorithm takes the frame number and shift everything from that frame till end of array to left and put it to the end of array. The miss rate of LRU is the second worst one in the table. LRU only cares about the past of address accesses. It predicts that if address is recently used, it will be probably used again soon. So it takes the least used one and evict it. In the implementation of CLOCK algorithm, I have added new ref attribute to each frame and new variable rec(clockhand). When new addr is inserted or hit happens it makes ref 1. If we need to evict a page. It starts from clockhand and iterates until find the frame where ref is equal to 0 and evict it, if it cant find it removes the clockhand and moves clockhand to next frame. From

the table we can see that CLOCK algorithm is better than FIFO and LRU. CLOCK algorithm acts like LRU, but with one difference it doesnt iterate the all frames and find the least recently used one, it starts where it is left after the previous eviction. It gives the second chance to addresses those are newly inserted. It increases the performance. In the implementation of OPT algorithm, firstly I read the trace file and make a linked-list to find the frame to evict. I dont add anything to linked list until coremap is full, because eviction happens when coremap is full. So I need addresses after that. After that in every access I remove the address from the linked-list, because we need the future not the past. If miss happens and program needs to evict the frame, OPT algorithm iterates the linked-list to find the frame that will be used lastest and evict it, if the frame will be never used then evict it. We can see that in most cases the OPT is better than others, because it knows the future and doesnt evict the pages that will be used very soon.

LRU algorithm is similar to OPT but unlike that it looks at the past. It works very well at the programs which exhibits the temporal locality. We can see from the table when the size of memory increases the miss-rate of LRU decreases. This happens because if we have more memory size we can make more prediction about the future. Let's examine the datas we got in 3 different traces. In simpleloop we see there is a difference between memsize 50-100 and in 150-200 it stays the same. In matmul-100 there is a really big difference between the 100-150, we can say that program exhibits more temporal locality with memsize 150 rather than 100. The reason for that change might be 150 addresses are being accessed very frequently and when program has memsize 100 approximately 50 addresses that will be accessed very soon will be evicted, because of the limited space, it will increase the page fault. In blocked-100-25 there is a change in 50-100 and after that it slightly changes, so the programs mostly exhibits the same locality with memsize 150-200

Task 3

Let's say we want to find belady anomaly with memsize n and n+1. Firstly, the pattern I found is n+1 different addresses and then first half of the addresses and then one new q address and again all first n+1 addresses and then q and first half except first one and again first n+1 addresses and then q and then first half except first and second one… It continues until q + first half except everything in first half, basically just q. In my trace hit rate is 52.6480 with memsize 20 and is 31.4642 with memsize 21. The anomaly happens with memsize 21 when the address we want to insert is evicted in previous eviction. So that cause to increase in the miss-rate.