Algorithm Design

- Here we first pad the sequence to make it of perfect power of 2, which will be helpful in computing complete binary tree.
- Next compute a prefix sum as explained in class with OP being (A+B)%MAX.
- **Prefix sum:**We compute 2 trees
    - *B* -> Data Flow up
        - We start with the data at the leaves of the tree and compute the parent at each level.
        - Where the root is the result of OP on the child nodes.
    - *C*-> Data Flow down
        - Here we start by copying the root of tree *B* computed above to the
        - This is then used in next level to compute its children.
        - if the node index is 0 we copy corresponding value from tree *B*
        - if the node index is odd we copy value from previous level
        - if the node index is even we perform operartion on C[h+1][(i/2-1)] OP B[h][i]
        - We get the prefix array at the leaves of tree C
- **Sorting**:
    - Now we use bitonic sorting to sort the prefix sum array computed above at leaves of tree C.
    - As we do swaping in prefix sum array, we also perform the swapping on the original array.
    - In the end we bring the data from device to host using memcpy.

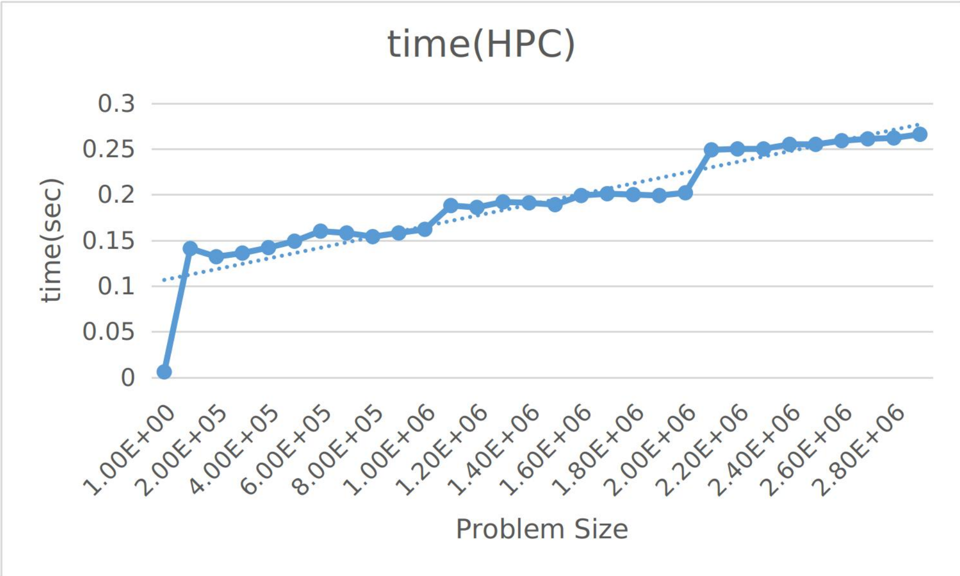
Timing Analysis

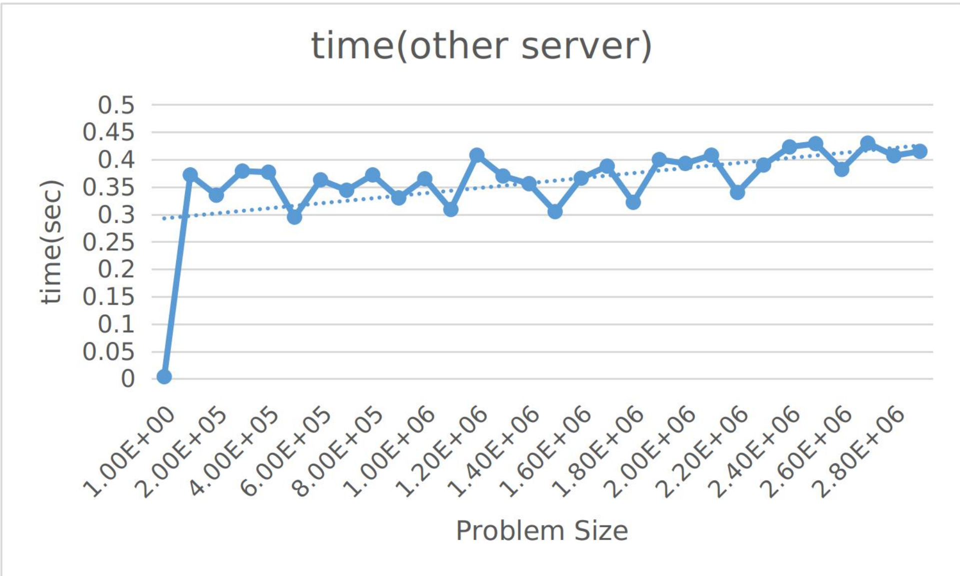
We do timing analysis on 2 systems, 1^{st} is HPC(GPU memory 12G) and second one is another server with higher GPU memory(32G).

- Small Size
- Large size
- Double size at every data point

Small size

- We see that when the problem size is very small, the time increases very slowly.
- In both the charts we see that there is a significant difference when we move from datasize 1 to any other data size, this is because in 1 size we are not using GPU, this indicates a setup time that adds up when we are using a GPU. This time is low(0.13) in HPC and higher in the other machine(0.3 sec).

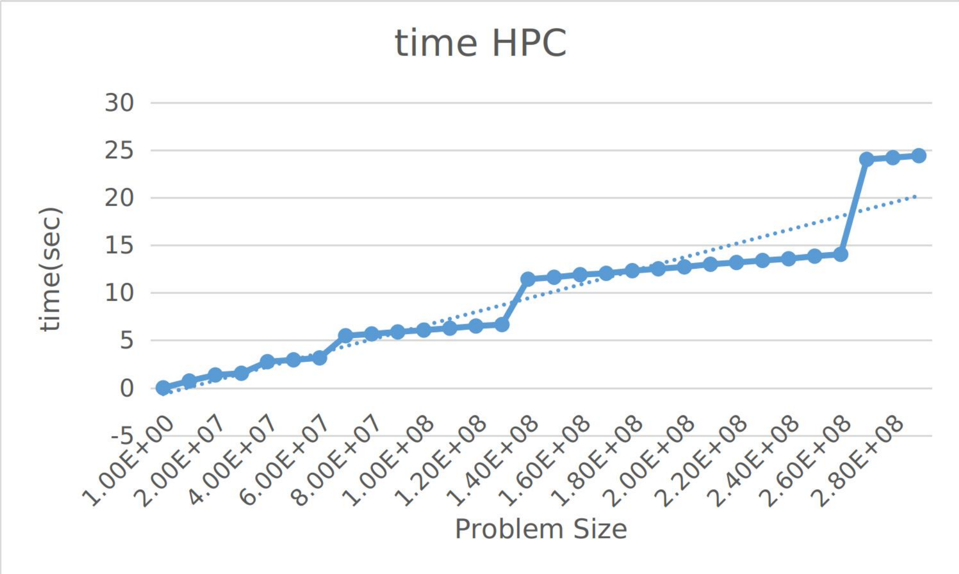


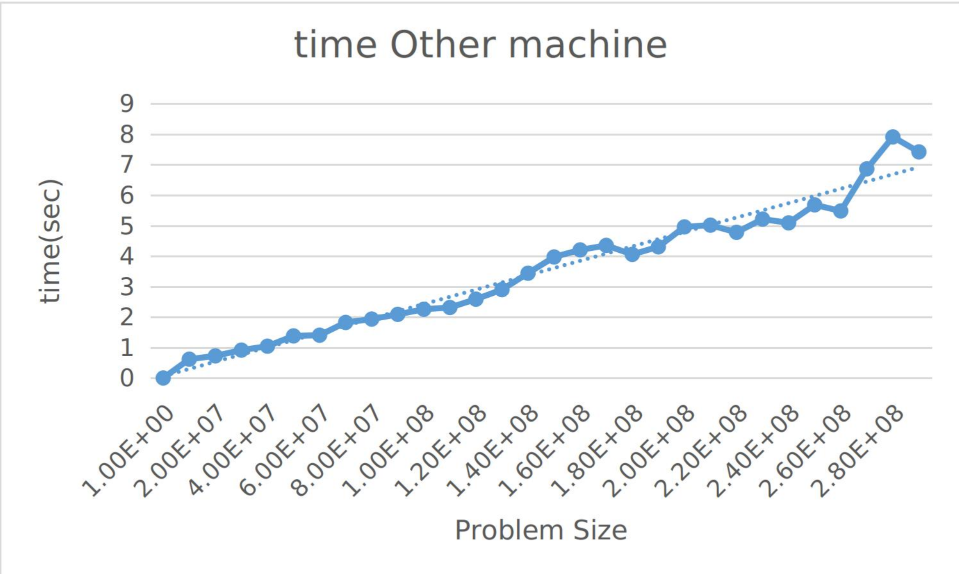


size	log size	time(HPC)	time(other)
1	0	0.006	0.004
100001	16.6096549	0.141	0.372
200001	17.60964769	0.132	0.335
300001	18.19460778	0.136	0.379
400001	18.60964408	0.142	0.377
500001	18.93157145	0.149	0.295
600001	19.19460538	0.16	0.363
700001	19.41699746	0.158	0.344
800001	19.60964228	0.154	0.372
900001	19.77956708	0.158	0.33
1000001	19.93157001	0.162	0.365
1100001	20.0690734	0.188	0.309
1200001	20.19460418	0.186	0.408
1300001	20.3100813	0.192	0.37
1400001	20.41699643	0.191	0.356
1500001	20.51653203	0.189	0.305
1600001	20.60964138	0.199	0.366
1700001	20.69710416	0.201	0.388
1800001	20.77956628	0.2	0.322
1900001	20.85756875	0.199	0.4
2000001	20.93156929	0.202	0.393
2100001	21.00195858	0.249	0.408
2200001	21.06907275	0.25	0.34
2300001	21.13320306	0.25	0.39
2400001	21.19460358	0.255	0.423
2500001	21.25349724	0.255	0.429
2600001	21.31008075	0.259	0.382
2700001	21.36452851	0.261	0.43
2800001	21.41699591	0.262	0.407
2900001	21.46762197	0.266	0.415

Large Size

- On a large problem size we see prominent stairs/steps. These steps are where the log size crosses integer numbers, this is because of the way the algorithm works by padding the extra 0s to make the problem size a power of 2.
- This is more prominent on HPC because on the other machine, the time difference is smaller on increasing size of problem, this may be because of presense of higher number of cores there.

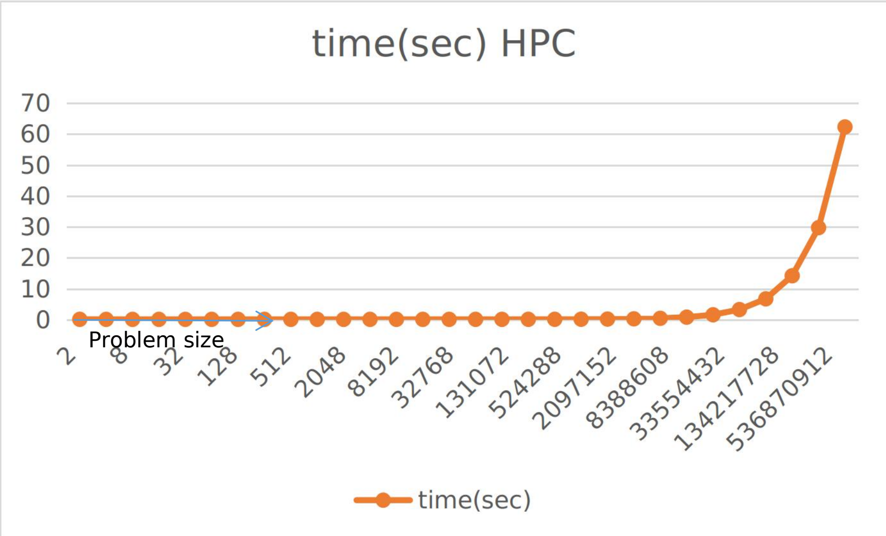


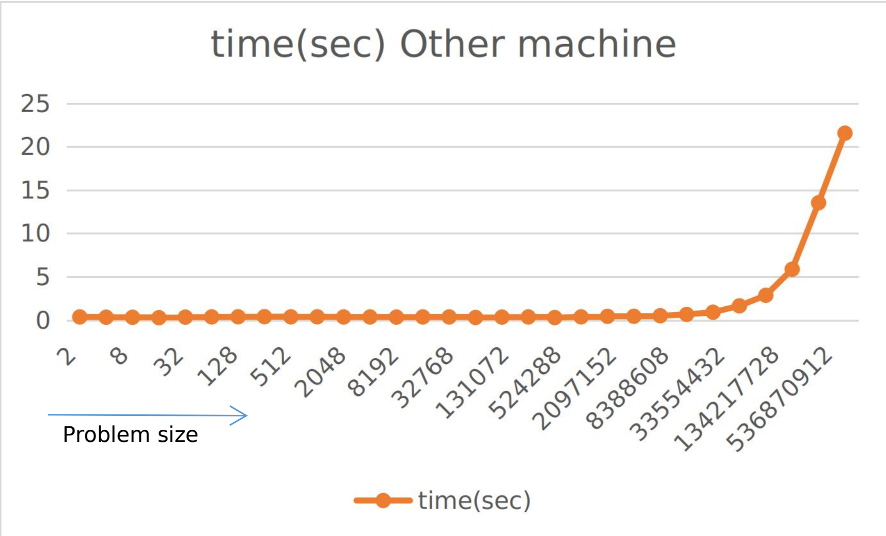


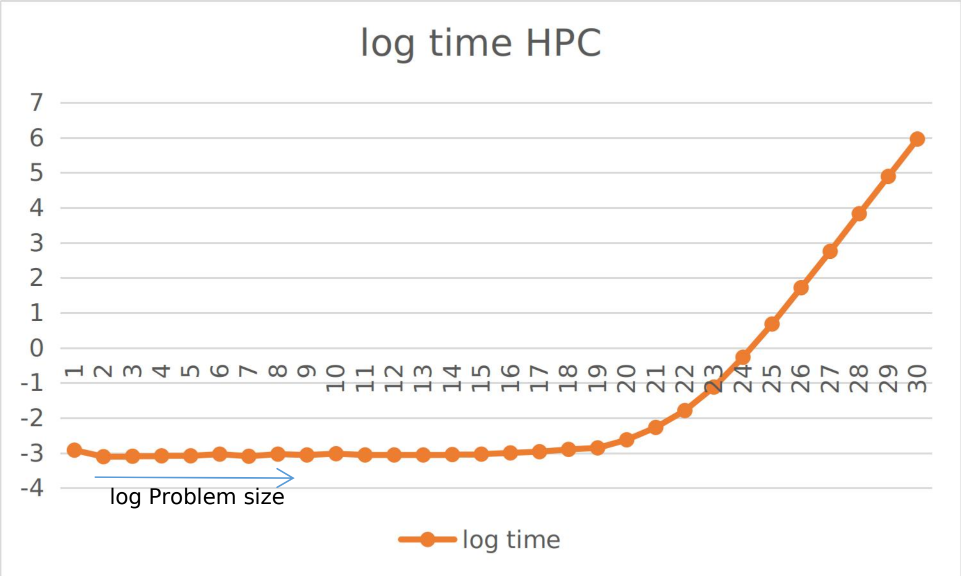
size	log size	time(HPC)	time(other)
1	0	0.003	0.004
10000001	23.25349681	0.72	0.622
20000001	24.25349674	1.356	0.73
30000001	24.83845921	1.533	0.919
40000001	25.2534967	2.751	1.048
50000001	25.57542479	2.947	1.385
60000001	25.83845919	3.15	1.408
70000001	26.06085161	5.483	1.826
80000001	26.25349668	5.675	1.937
90000001	26.42342168	5.883	2.092
100000001	26.57542477	6.074	2.259
110000001	26.7129283	6.274	2.317
120000001	26.83845918	6.504	2.59
130000001	26.95393639	6.657	2.903
140000001	27.0608516	11.431	3.442
150000001	27.16038727	11.635	3.974
160000001	27.25349667	11.906	4.206
170000001	27.34095951	12.047	4.354
180000001	27.42342167	12.328	4.059
190000001	27.50142419	12.525	4.307
200000001	27.57542477	12.726	4.961
210000001	27.64581409	13.002	5.018
220000001	27.71292829	13.188	4.782
230000001	27.77705863	13.402	5.217
240000001	27.83845917	13.58	5.093
250000001	27.89735286	13.853	5.684
260000001	27.95393639	14.05	5.483
270000001	28.00838417	24.038	6.865
280000001	28.06085159	24.219	7.913
290000001	28.11147766	24.427	7.421

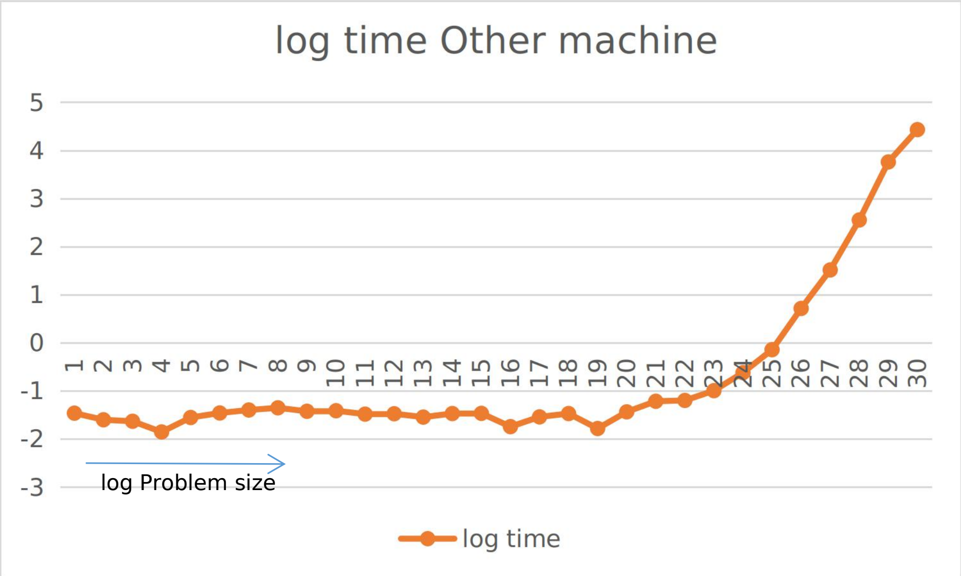
Double size at every data point

- Here we try running the algo on different sizes; doubling it on every next point.
- Here we see much smoother graph without the steps,
- Also, we observe that log graph becomes linear at the end when the setup time becomes negligible






log time HPC
7
6
5
4
3
2
1
0
-1
-2
-3
-4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
log Problem size
log time


log time Other machine
5
4
3
2
1
0
-1
-2
-3
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
log Problem size
log time

size	log size	time(other)	log time	time(HPC)	log time
2	1	0.363	-1.46	0.132	-2.92
4	2	0.33	-1.60	0.116	-3.11
8	3	0.323	-1.63	0.117	-3.10
16	4	0.277	-1.85	0.118	-3.08
32	5	0.341	-1.55	0.118	-3.08
64	6	0.364	-1.46	0.122	-3.04
128	7	0.38	-1.40	0.117	-3.10
256	8	0.392	-1.35	0.122	-3.04
512	9	0.373	-1.42	0.12	-3.06
1024	10	0.376	-1.41	0.123	-3.02
2048	11	0.358	-1.48	0.12	-3.06
4096	12	0.36	-1.47	0.12	-3.06
8192	13	0.343	-1.54	0.12	-3.06
16384	14	0.361	-1.47	0.121	-3.05
32768	15	0.362	-1.47	0.122	-3.04
65536	16	0.299	-1.74	0.125	-3.00
131072	17	0.344	-1.54	0.128	-2.97
262144	18	0.361	-1.47	0.134	-2.90
524288	19	0.291	-1.78	0.138	-2.86
1048576	20	0.37	-1.43	0.162	-2.63
2097152	21	0.431	-1.21	0.207	-2.27
4194304	22	0.436	-1.20	0.289	-1.79
8388608	23	0.502	-0.99	0.46	-1.12
16777216	24	0.651	-0.62	0.83	-0.27
33554432	25	0.906	-0.14	1.602	0.68
67108864	26	1.641	0.71	3.287	1.72
134217728	27	2.857	1.51	6.751	2.76
268435456	28	5.867	2.55	14.212	3.83
536870912	29	13.539	3.76	29.75	4.89
1073741824	30	21.572	4.43	62.299	5.96