**1. Explain briefly how the Karmarkar-Karp algorithm can be implemented in O(nlogn) steps, assuming**

the values in A are small enough that arithmetic operations take one step.

If we load input data in a max heap, we can build it in O(nlog(n)), at each operation, it requires max-heapify which lead to O(log(n)), so total time complexity still O(nlog(n))

**2. Numerical results, and the time taken by the algorithms.**

KK: Karmarkar-Karp

RR Stan: Repeated Random with standard representation

RR pp: Repeated Random with Pre-partitioning

HC climb: Hill climbing with standard representation

HC pp: Hill climbing with Pre-partitioning

SA climb: Simulated annealing with standard representation

SA pp: Simulated annealing with Pre-partitioning

Iterations = 25000

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | KK | RR Stan | RR pp | HC stan | Hc pp | SA stan | SA pp |
| residue  (average) | 196960 | 218383699 | 167 | 289044475 | 1146 | 305675256 | 222 |
| time[s]  (average) | 0.0002 | 1.12 | 12.01 | 0.211 | 10.54 | 0.84 | 14.2 |

Iterations = 2500

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | KK | RR Stan | RR pp | HC stan | Hc pp | SA stan | SA pp |
| residue  (average) | 288992 | 3,258,467,639 | 1586 | 1,433,526,477 | 3334 | 1,514,535,331 | 1812 |
| time[s]  (average) | 0.0002 | 0.102 | 1.18 | 0.02 | 1.07 | 0.08 | 1.41 |

**3. Compare the results and discussion**

**- Compare the results**

1. Pre-partitioning led to solutions with dramatically lower residues, when compared to standard  
representation.  
2. In no case did a randomized heuristic with standard representation return results lower than, or  
even comparable to, those returned by Karmarkar-Karp.

3. Using pre-partitioning, all three randomized heuristics gave substantially better solutions than  
did Karmarkar-Karp.

4. Pre-partitioning led to substantially longer runtimes for all three randomized heuristics.  
5. In the Pre-partitioned representation, increasing the number of iterations improved results from  
repeated-random and simulated annealing more than it did for hill-climbing.

**- Discussion**

The single best performance in both cases, in terms of residue, was repeated-random with Pre-partitioning. In the case of 25,000 iterations, however, this method also took the longest.

Meanwhile, simulated annealing with pre-partitioning gave results within 150% of repeated-random, while taking about 40% more runtime. At lower iterations, however, repeated-random did not take significantly longer than the other two randomized heuristics, and still gave the best results. Karmarkar-Karp, meanwhile, was in both cases the fastest method – by one to several orders of magnitude.

These results illustrate some interesting trade-offs among the various heuristics and representations.

If one wants a solution very quickly and only requires that it be a fairly good one, Karmarkar-Karp may be the best choice.

If better solutions are needed and there is more time to trade off, a randomized heuristic with pre-partitioning may be more useful.

If one needs very good solutions and does not mind waiting a long time for them, a high iteration implementation of either repeated random or simulated annealing (both with pre-partitioning) may be the best tool to use

**4. How you could use the solution from the Karmarkar-Karp algorithm as a starting point  
for the randomized algorithms**

- Standard Representation

After apply Karmarkar-Karp algorithm, we can get the partition result S0.

Then we can start search neighboarhood starting from this S0.

- Pre-Partioning

After apply Karmarkar-Karp algorithm, we can get the partition result S0.

We can generate initial partition P0 can be obtained by initially assigning pi to 1 for all Si = -1 and similarly assigning pi to 2 for Si = 1

In this case, we can get the solution faster than original one.