CS 124 Programming Assignment 3: Spring 2023

Your name(s) Yuting Liu:

Collaborators: (You shouldn't have any collaborators but the up-to-two of you, but tell us if you did.)

No. of late days used on previous psets: 8
No. of late days used after including this pset: 9

1. Dynamic programming solution to the Number Partition problem

We can design the number partition problem as a dynamic programming problem: We have a sequence A including numbers $a_1, a_2, ..., a_n$ and we need to separate them into two subset A_1, A_2 so that the subset sum of either A_1 or A_2 should be the closest to $\lfloor \frac{b}{2} \rfloor$, where the sequence of terms in A sum up to some number b.

1. Define D[i,j] as true if the sum of a subset combination among $a_1, a_2, ..., a_j$ is i.

2. We can solve the problem by solving its subproblems. D[i,j] is true if D[i,j-1] is true (when a_j is in another subset) or D$[i - a_j, j - 1]$ is true (when a_j is in this subset).

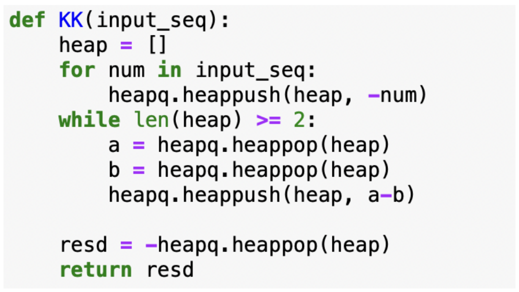
3. By letting i from 1 to $\lfloor \frac{b}{2} \rfloor$ when j from 1 to n, we can build all instances of D[i,j]. The answer table have the size of $\lfloor \frac{b}{2} \rfloor$ by n.

4. The actual solution is the true D[i,j] true with the largest i.

This DP algorithm is polynomial in n and b. The two loops takes n*$\lfloor \frac{b}{2} \rfloor$ steps and in each step we need two check in the answer table and one value assignment in the answer table, the running time is $O(n * \lfloor \frac{b}{2} \rfloor) \equiv n * b$.

2. Karmarkar-Karp running time

The running time of KK algorithm is O(nlogn). At first we need to insert all elements into a priority queue, which takes O(nlogn) at worst for n elements. The inserted elements are multiplied by -1, taking constant time. Then we pop out two elements taking O(logn), subtract them taking O(1) as assumptions, and add the result into the heap taking O(logn). For each step the heap shorten by 1, so until the heap contains only 1 element, there are n-1 such steps taking O((n-1)logn). Finally we pop out the last element in this heap, taking O(logn). Adding them up, the KK algorithm takes O(nlogn) when assuming all arithmetic operations take one step. This code is demonstrated as follows:



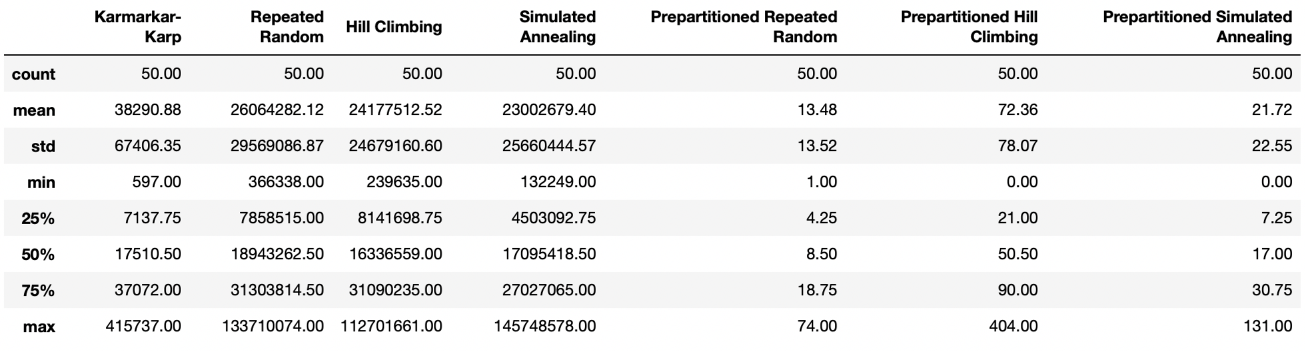
```python
def KK(input_seq):
    heap = []
    for num in input_seq:
        heapq.heappush(heap, -num)
    while len(heap) >= 2:
        a = heapq.heappop(heap)
        b = heapq.heappop(heap)
        heapq.heappush(heap, a-b)

    resd = -heapq.heappop(heap)
    return resd
```

Figure 1: Karmarkar-Karp Algorithm

Discussion

Residue summary

I generated 50 random instances of the number partition problem, and for each instance seven method to acquire the residues were applied. For simulated annealing algorithm I applied $T(iter) = 1010(0.8)\lfloor\frac{iter}{300}\rfloor$. Each instance was created with 100 random integers ranging from 0 to 10^{11}. The results of 50 instances are shown below:



	Karmarkar-Karp	Repeated Random	Hill Climbing	Simulated Annealing	Prepartitioned Repeated Random	Prepartitioned Hill Climbing	Prepartitioned Simulated Annealing
count	50.00	50.00	50.00	50.00	50.00	50.00	50.00
mean	38290.88	26064282.12	24177512.52	23002679.40	13.48	72.36	21.72
std	67406.35	29569086.87	24679160.60	25660444.57	13.52	78.07	22.55
min	597.00	366338.00	239635.00	132249.00	1.00	0.00	0.00
25%	7137.75	7858515.00	8141698.75	4503092.75	4.25	21.00	7.25
50%	17510.50	18943262.50	16336559.00	17095418.50	8.50	50.50	17.00
75%	37072.00	31303814.50	31090235.00	27027065.00	18.75	90.00	30.75
max	415737.00	133710074.00	112701661.00	145748578.00	74.00	404.00	131.00

Figure 2: Residue Description

According to this chart, KK generates residues much smaller than Repeated Random, Hill Climbing, and Simulated Annealing method, but much higher than Prepartitioned Repeated Random, Prepartitioned Hill Climbing, and Prepartitioned Simulated Annealing method. The prepartitioned methods have sigfinicantly better performance.

Among the prepartitioned methods, Prepartitioned Repeated Random method has the best performance with the lowest mean and the lowest std. Prepartitioned Hill Climbing method is the worst with the largest mean and the largest std.

Among the standard methods, simulated annealing is the best with the lowest mean and the lowest std. Repeated random method is the worst with the largest mean and the largest std.

The distribution curve of such 7 methods are demonstrated as follows. The x scale is $log(residue + 1)$ to compress all seven distributions and avoid nan for zero residues.

The running time for each iteration for these methods are demonstrated as follows. The KK algorithm is the fastest method among all, and the standard representation methods are much faster than the prepartitioned methods. The preparitioned methods are slower because they need to prepartioned the original sequence, and also calculate the residue using KK algorithm. In addition, the simulated annealing method of both representations are the slowest as they need to compare the residue twice, and compute a annealing function as the probability. Besides, the hill climbing method of both representations are the fastest except the KK algorithm, which means looking for a random neighbor of a solution is faster than looking for another random solution.

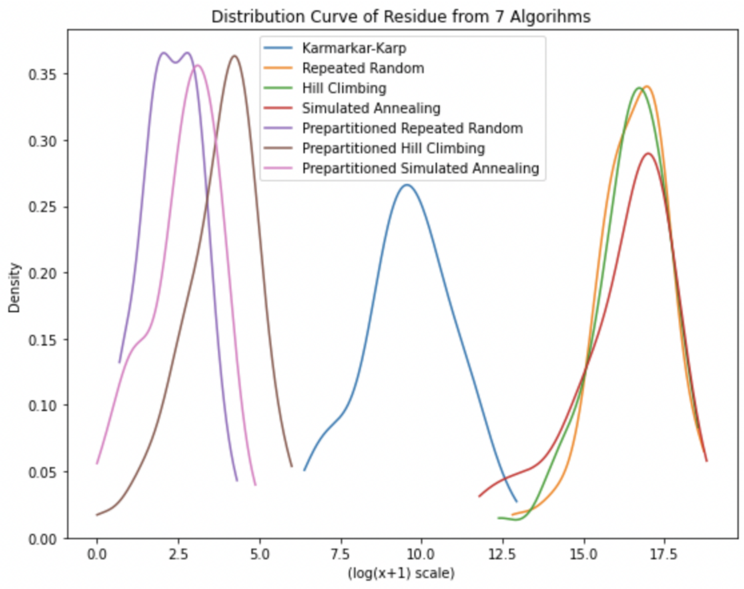


Figure 3: Residue Distribution

Running time	
Method name	Running time per iteration (ms)
Karmarkar-Karp	6.561279296875e-06
Repeated Random	0.07369256019592285
Hill Climbing	0.054038438796997074
Simulated Annealing	0.09566797256469727
Prepartitioned Repeated Random	0.6823272323608398
Prepartitioned Hill Climbing	0.6544557666778564
Prepartitioned Simulated Annealing	1.753415174484253

KK as the starting point

One way to use the solution from the Karmarkar-Karp algorithm as a starting point for randomized algorithms is to use it as the initial solution for hill climbing or simulated annealing algorithm. The idea is to use the deterministic solution from Karmarkar-Karp to find a good starting point for the randomized algorithm, which can then explore the solution space to potentially find better solutions.

Using the Karmarkar-Karp solution as a starting point for randomized algorithms may have the effect of improving the quality of the solutions found by the randomized algorithms. For instance, in hill climbing algorithm, we can take the initial solution as the solution of KK, and take the local search of KK solution's neighbors. This is beneficial as the residue of the final solution must become no more than the KK solution, which improves the performance of the hill climbing method significantly, and also saving steps to find a local optimal solution. In addition, we might also use the KK solution for initialization in the simulated

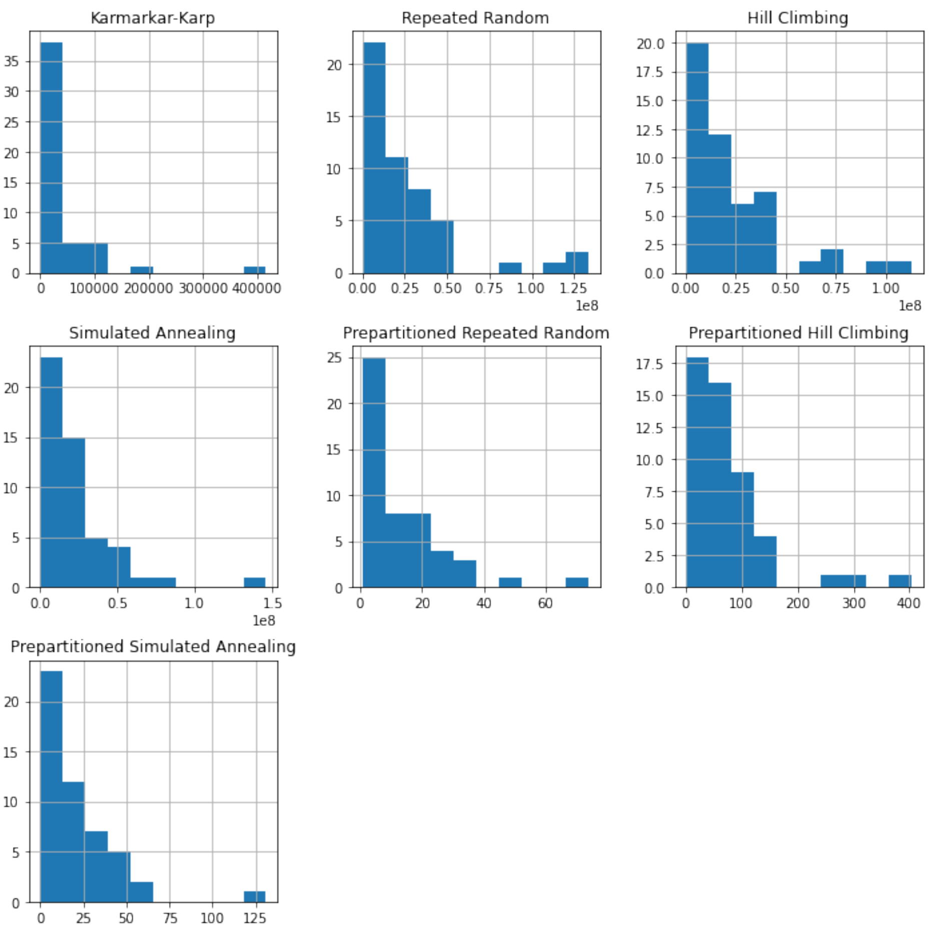


Figure 4: Residue Histogram

annealing method. However, this is not so effectively beneficial as it is placed in the hill climbing, as simulated annealing method not always move to the better neighbor. Using in the repeated random won't lead to the better solution as it compares residues of two independent random solutions. The disadvantage of using the KK solution as the starting point is that without random initialization it is prone to be trapped and get the local optimal solution rather than a global optimal solution.