CS 120: Intro to Algorithms and their Limitations

Problem Set 4

Due: October 18, 2023 11:59pm **Denny Cao**

Collaborators:
No. of late days used on previous psets: 1
No. of late days used after including this pset: 2

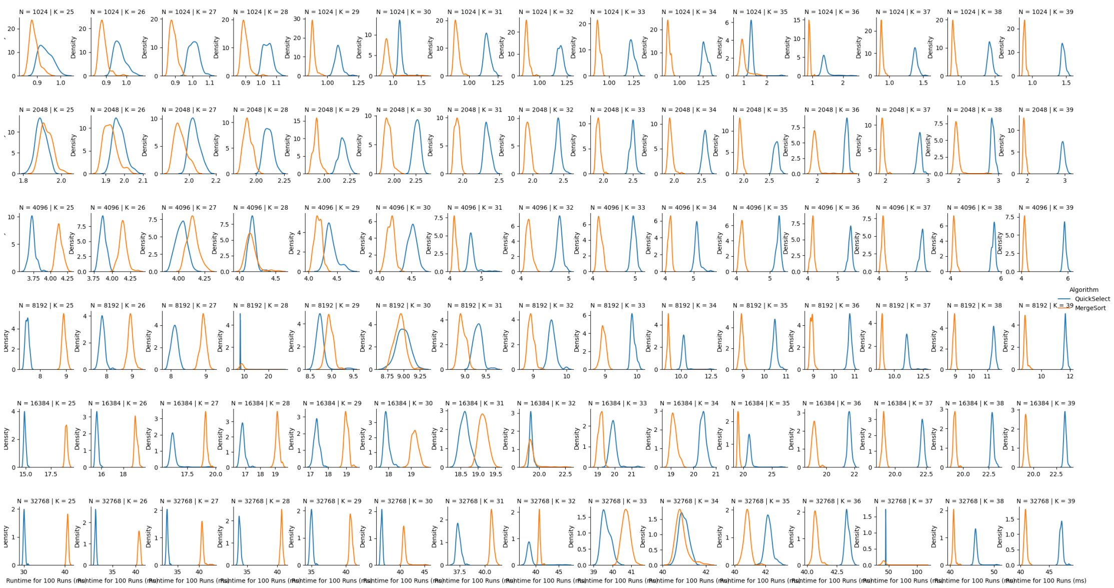
§1 Randomized Algorithms in Practice

(a) In `ps_4.py`.

(b) The table below depicts corresponding values of k for each n:

n	k_n^*
1024	29
2048	29
4096	31
8192	33
16384	34
32768	36

From running the experiment for $25 \leq k \leq 39$, we obtain the following graphs for each n:

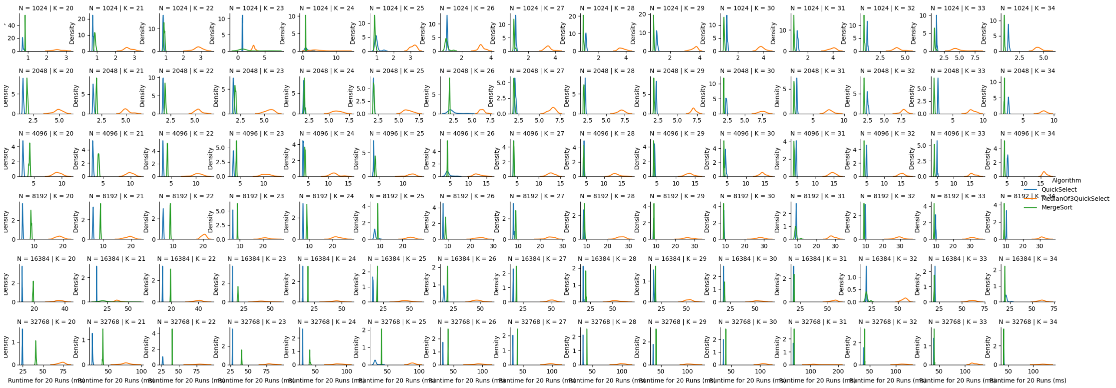


We select the least value k for each n when the distribution of runtimes for `MergeSortSelect` is less than the distribution of runtimes for `QuickSelect`, as this would imply that, for that chosen k, `MergeSortSelect` is faster than `QuickSelect`, it is the first k such that `MergeSortSelect` is the faster algorithm on average than `QuickSelect`.

(c) A functional form for k_n^* could be the following:

$$k^*(n) = \log_2 n + 20$$

(d)



The expected result is that, by selecting the pivot position with less randomness, we can have more "even" splits of the array, decreasing the chance that `QuickSelect` reaches the $O(n^2)$ time.

§2 Dictionaries and Hash Tables

Claim — `DuplicateSearch` can be solved by a Las Vegas algorithm with expected runtime $O(n)$ using a dictionary data structure.

Proof. We first describe the Las Vegas algorithm:

1. `Preprocess(`$\mathbb{R}$`,`m`)`: Initialize an array A of size m, where $m = \Theta(n)$. Choose a random hash function $h : \mathbb{R} \to [m]$.
2. We will now loop through all elements (K, V) of the input array and do the following:
   a) `Search(`K`)`: Walk through the linked list $A[h(K)]$. If search returns an element, we return K. If search returns $\perp$, continue.
   b) `Insert(`K,V`)`: Insert (K, V) at the head of linked list slot $A[h(K)]$.
3. If the loop is completed, return $\perp$.

We now want to prove our algorithm:

1. has the desired runtime and
2. is correct.

- Initializing an array of size $O(n)$ will take $O(n)$ time. For each element (K, V) in the input array, `Search(`K`)` will take an expected time of $O(1)$, as `Search(`K`)` takes $O(1 + \frac{n}{m})$ and $m = \Theta(n) \implies m = \Omega(n)$. `Insert(`$K,V$`)` will also take $O(1)$, as we simply insert the element at the head of the linked list at $A[h(K)]$. At the worst case, we iterate through all elements of the input array, and, as we take an expected time of $O(1)$ time for n elements, it will expectedly take $O(n)$ time. Thus, in total, the expected run time is the time for preprocessing and the expected time it takes for the loop: $O(n) + O(n) = O(n)$.

- Before inserting a key value pair (K, V), we run `Search(`K`)` which will only return if the dictionary contains a key value pair (K, V^*). The only values in the dictionary are key value pairs from the input that were inserted when `Search(`K`)` returns $\perp$; `Search(`K`)` will only return K, causing the algorithm to return K when there exists a duplicate, and the loop will complete only if for all (K, V) in the input array, `Search(`K`)` returns $\perp$ prior to inserting, causing the algorithm to return $\perp$, meaning there is no duplicate present.

We have shown that there exists a Las Vegas algorithm with expected runtime $O(n)$ using a dictionary structure and that the algorithm is correct. □

§3 Rotating Walks

(a) *Proof.* We first describe the reduction algorithm:

a) `Preprocess`: We preprocess a new digraph G' as follows:
   - Define the vertex set V' of G' to be the set of pairs (v, j), where $v \in V$ is a vertex from the original digraph, and $j \in [k]$
   - For each pair of vertices (v, j) and $(w, j + 1 \bmod k)$ in V', if there is an edge (v, w) in $E_{j+1 \bmod k}$, then add an edge from (v, j) to $(w, j + 1 \bmod k)$ in E', the edge set of G'.

b) Make an oracle call `SingleSourceShortestPaths(`$G', (s, 0)$`)`.

c) `Postprocess`: As the oracle returns an array of shortest paths from s to all other vertices $v \in V'$, we iterate to find the shortest path amongst $(t, j) \forall j \in [k]$; we find:

$$\texttt{dist}_{G'}((s, 0), (t, x)) \leq \texttt{dist}_{G'}((s, 0), (t, j)) \forall j \in [k], x \in [k]$$

d) Return the path from $(s, 0)$ to (t, x).

We now want to prove our algorithm:

a) has the desired runtime and

b) is correct.

- Preprocessing G' takes $O(kn)$, as we iterate through all n vertices for each of the k diagraphs. `SingleSourceShortestPaths` can be solved in $O(n+m)$, where n is the number of vertices and m is the number of edges. As we call the oracle on G' which has a vertex set $V' = \{(v_0, 0), \ldots, (v_0, k-1), \ldots, (v_{n-1}, 0), (v_{n-1}, k-1)\}$ and thus $|V| = kn$ and an edge set E' where $|E'| = |E_0| + |E_1| + \cdots + |E_{k-1}|$, $|E'| = m_0 + m_1 + \cdots + m_{k-1}$, `SingleSourceShortestPaths` on G' can be solved in time $O(|V'| + |E'|) = O(kn + m_0 + m_1 + \cdots + m_n)$. The time it takes during postprocessing to iterate through all shortest paths to find the shortest path from $(s, 0)$ to (t, x) takes $O(kn)$. Thus, the total runtime is $O(kn) + O(kn + m_0 + m_1 + \cdots + m_{k-1}) + O(kn) = O(kn + m_0 + m_1 + \cdots + m_{k-1})$.
- As each edge in G' is of the form $((v_i, j), (v_t, j + 1 \mod k))$, each edge is a rotation from G_j to $G_{j+1 \mod k}$. We can thus use `SingleSourceShortestPaths` to find the shortest path from any vertex to another formed by any rotated path.

We have shown that the problem of `ShortestRotatingWalk` from s to t with respect to $G_0, \ldots, G_{k-1}$ can be reduced to `SingleSourceShorestPaths` with the desired runtime and that the algorithm is correct. $\square$

(b)

d	Frontier F_d	Predecessor Relationships
0	$\{(a,0)\}$	
1	$\{(b,0)\}$	$((a,0),(b,0))$
2	$\{(d,1),(e,1)\}$	$((b,0),(d,1)),((b,0),(e,1))$
3	$\{(d,0),(g,0)\}$	$((e,1),(d,0)),((e,1),(g,0))$
4	$\{(f,1),(c,1)\}$	$((d,0),(c,1)),((d,0),(f,1),((g,0),(f,1))$

(c) *Proof.* Let vertices be denoted by v_{ij} where i is the row and j is the column. Let G_0, G_1, G_2 denote the graphs of Player 0, Player 1, and Player 2 respectively and the possible positions they can move.

- G_0, since Player 0 can move like a chess rook, Player 0 can move from a vertex v_{ab} to a vertex $v_{a,x}$ or $v_{x,b}$ where $0 \leq x \leq n$. Each vertex can move to $2n-2$ other vertices, and as there are n^2 vertices,$|E_0| = n^2(2n-2) = O(n^3)$.
- G_1, since Player 1 can move like a chess bishop and the greatest amount of outward edges for a vertex is $2n-2$, we know $|E_1| \leq n^2(2n-2) = O(n^3)$.
- G_2, since Player 2 can move like a knight which has at most 8 outward edges for a vertex, $|E_2| \leq 8n^2 = O(n^2)$.

We now call the oracle `ShortestRotatingPaths` with the graphs G_0, G_1, G_2 and start position s and desired position t, which will have runtime $O(|V| + |E|)$. As $|V| = n^2$ and $|E| = |E_0| + |E_1| + |E_2| = O(n^3) + O(n^3) + O(n^2) = O(n^3)$, the runtime will be $O(n^2 + n^3) = O(n^3)$, as desired. $\square$