COMPSCI 260P PROJECT #1 — MEDIAN SELECTION

QUICKSELECT AND ITS COMPLEXITY

- Quick-select is a randomized selection algorithm based on the prune-and-search paradigm. It picks a random element x as pivot and partitions input set into 3 sequences: L: containing elements lesser than x, E: with elements equal to x and G: containing elements greater than x.
- Depending on k, either the result is in E or we need to recur in L or G.
- Randomized quick-select runs in **O(n)** expected time, taken over all possible random choices made by the algorithm, and this expectation does not depend whatsoever on any randomness assumptions about the input distribution.

Pseudocode

Algorithm quickSelect(S, k):
Input: Sequence S of n comparable elements, and an integer $k \in [1, n]$
Output: The kth smallest element of S
if n = 1 then
    return the (first) element of S
pick a random element x of S
remove all the elements from S and put them into three sequences:
- L, storing the elements in S less than x
- E, storing the elements in S equal to x
- G, storing the elements in S greater than x.

if $k \leq |L|$ then
    quickSelect(L, k)
else if $k \leq |L| + |E|$ then
    return x // each element in E is equal to x
else
    quickSelect(G, $k - |L| - |E|$)

Theoretical complexity

- The choice of pivot is important for determining expected running time. **1/2** of possible pivots (middle of a sequence S) cause good calls. The size of L and G should be less than **3s/4** for good calls and one of L and G should be greater than 3s/4 for bad call.
- In this project, rand() function has been used to generate the pivot element index and the worst-case scenario doesn't occur for this and the performance is O(n).
- Worst case scenario is observed by picking the smallest and largest element (0 and n-1) as pivot, for a set of n elements. The performance is $O(n^2)$.
- Theoretically, justifying the claim that quick select runs in O(n) time requires the simplest of probabilistic arguments. The main probabilistic facts that we use is the linearity of expectation.

Probabilistic fact 1: Expected number of coin tosses for head is two.

Probabilistic fact 2: Expectation is linear function.

$E(X+Y) = E(X) + E(Y)$

$E(cX) = cE(X)$

Best, Average Case

Let T(n) denote expected running time of quick select. From facts 1 & 2, we get,

⇨ $T(n) \leq T(3n/4) + bn*$(expected no. of calls before good call)

⇨ $T(n) \leq T(3n/4) + 2bn$

Converting this equation into closed form, after 2 iterative applications we get, T(n) is geometric series,

⇨ $T(n) \leq T((3/4)^2 n) + 2b(3/4)n + 2bn$

⇨ $T(n) \leq 2bn + 2b(3/4)n + 2b(3/4)^2 n) + \ldots$

⇨ $T(n) = O(n)$

Thus, we can solve quick select selection problem in **O(n)** expected time.

Worst Case

The worst case happens when the pivot is not in desired location (middle of a sequence S). The pivot is either the smallest or largest element in the input sequence S. In this case, the recursion is performed on a list one item smaller than the previous list.

⇨ $T(n) = 1 + n + T(n - 1)$

⇨ $T(n) = n + (n - 1) + (n - 2) + \cdots + 2 + 1$

⇨ $n(n - 1)/ 2$

⇨ $O(n^2)$

Thus, worst case can solve quick select selection problem in $\mathbf{O(n^2)}$ expected time.

Design choices (data structure)

- The implementation for this algorithm is done in C++. The input sequence S is in the form of an integer Vector. Initially the integer Vector is filled with N distinct elements using in-built generate() method dynamically.
- The index of the pivot element is found using the rand() function of C++ in order to randomize the pivot selection X. After finding X, the input elements are put into 3 sequences of integer Vectors – Less, Equal, Greater.
- Based on the K value, recursive call is made by passing either Less or Greater vector to quick select function or the value Vector[randomMedian] is returned if K is equal |L|+|E|.

DSELECT

This selection algorithm that runs in worst case O(n) time. The pivot is chosen deterministically and not randomly.

Pseudocode

Algorithm DeterministicSelect(S, k):
Input: Sequence S of n comparable elements, an integer k $\in$ [1, n] and size $\in$ [3,5,7,9]
Output: The kth smallest element of S
if n = 1 then
    return the (first) element of S
Divide S into g = n/size groups, S1, . . . , Sg, such that each of groups S1, . . . , Sg–1 has 'size' elements and group Sg has at most 'size' elements.
for i ← 1 to g do
Find the baby median, xi, in Si (using any method)
    x ← DeterministicSelect({x1, . . . , xg}, g/2)
remove all the elements from S and put them into three sequences:

- L, storing the elements in S less than x
- E, storing the elements in S equal to x
- G, storing the elements in S greater than x.

if k ≤ |L| then
    DeterministicSelect(L, k)
else if k ≤ |L| + |E| then
    return x // each element in E is equal to x
else
    DeterministicSelect(G, k – |L| – |E|)

Design choices

- The implementation for this algorithm is done in C++. The input sequence S is in the form of an integer Vector. Initially the integer Vector is filled with N distinct elements using in-built generate() method dynamically.
- The index of the pivot element is found by dividing sequence S by 'N/size' groups of 'size' elements each where size = 3,5,7 or 9. Median of each group is found by iterating from 1 to 'size'. Recursively the median of baby median is found to find pivot X.
- After finding X, the input elements are put into 3 sequences of integer Vectors – Less, Equal, Greater.
- Based on the K value, recursive call is made by passing either Less or Greater vector to quick select function or the value Vector[randomMedian] is returned if K is equal |L|+|E|.

Theoretical analysis of asymptotic complexity of the variants of DSelect
Each call to DeterministicSelect takes O(n) time, not counting the recursive calls.

Analysis for Groups of Three

- The algorithm has 2 recursive calls, the first one is performed on a set of baby medians of size n/3,

    ⇨ g=n/3 →1

- The second recursive call is made either on set L or G based on pivot value X. Consider an example, where there is sequence of 3 distinct elements (N=3). After determining the median (2^{rd} element in sequence S) there would be at least **2** elements <= median i.e $2 \leq |L|+|E|$. We have that for g/2 => **(1/2)(n/3) groups**, at least half of the group elements are less than or equal to X.
- We conclude the second recursive call is performed in set of at most,
  - ⇨ n – 2[(1/2)(n/3) -2]
  - ⇨ n – (n/3 -4)
  - ⇨ 2n/3 + 4 →2
- Overall the running time T(n) is determined by combining the above 2 sub problems 1&2 along with O(n), which is work done apart from 2 recursions,
  - ⇨ $T(n) \leq T(n/3) + T(2n/3+4) + O(n)$ →3
- We guess,
  - ⇨ $T(n) \leq cn$, for some constant $c > 0$ →4
  - ⇨ $cn/3 + 2cn/3 + 4c + bn \leq cn$, for constant $b>0$ from 3&4
  - ⇨ $4c + bn \leq 0$
  - ⇨ $c \leq -bn/4$, which contradicts our assumption of $c > 0$ (from 4)
- Also the coefficients sum to 1/3 + 2/3 = 1, which is not < 1, hence recursion cannot be solved in linear time.
- We are still left with a sub problems of total size n. Thus, we did not manage to reduce the size of the problem effectively.
- The runtime for DSelect with group 3 would be **O(nlogn).**

Analysis for Groups of Five

- The algorithm has 2 recursive calls, the first one is performed on a set of baby medians of size n/5,
  - ⇨ g=n/5 →1
- The second recursive call is made either on set L or G based on pivot value X. Consider an example, where there is sequence of 5 distinct elements (N=5). After determining the median (3^{rd} element in sequence S) there would be at least **3** elements <= median i.e $3 \leq |L|+|E|$. We have that for g/2 => **(1/2)(n/5) groups**, at least half of the group elements are less than or equal to X.

- We conclude the second recursive call is performed in set of at most,
  - ⇨ n – 3[(1/2)(n/5) -2]
  - ⇨ n – (3n/10 -2)
  - ⇨ 7n/10 + 6 →2

- Overall the running time T(n) is determined by combining the above 2 sub problems 1&2 along with O(n), which is work done apart from 2 recursions,
  - ⇨ $T(n) \leq T(n/5) + T(7n/10+6) + O(n)$ →3

- The above recurrence equation 3 is solved by guessing that -

- ⇨ $T(n) \leq cn$, for some constant $c > 0$ →4
- ⇨ $T(n) \leq cn/5 + 7cn/10 + 6c + bn$, where $b>0$ is constant →5

- Solving above 2 equations, from 4 & 5 we get,
  - ⇨ $9cn/10 + 6c + bn \leq cn$ →6
  - ⇨ $bn + 6c \leq cn-(9cn/10)$
  - ⇨ $bn + 6c \leq cn/10$
  - ⇨ $cn/10 - 6c \geq bn$
  - ⇨ $c[(n/10)-6] \geq bn$
  - ⇨ $c \geq 10bn/[n-60]$

- We should choose $n>60$ and we should get constant $c>0$. If we choose $n = 2*60 = 120$, we get **$c \geq 20b$**, which satisfies $c>0$ for our guess $T(n) \leq cn$.
- Also, since the coefficients sum to $1/5 + 7/10 = 9/10 = 0.9 < 1$, recursion solves in linear time. Hence the DSelect algorithm with 5 grouping can be solved in linear time **O(n).**

Analysis for Groups of Seven

- The algorithm has 2 recursive calls, the first one is performed on a set of baby medians of size n/5,
  - ⇨ $g=n/7$ →1
- The second recursive call is made either on set L or G based on pivot value X. Consider an example, where there is sequence of 7 distinct elements (N=7). After determining the median (4^{rd} element in sequence S) there would be at least **4** elements <= median i.e $4 \leq |L|+|E|$. We have that for g/2 => **(1/2)(n/7) groups**, at least half of the group elements are less than or equal to X.

- We conclude the second recursive call is performed in set of at most,
  - ⇨ $n - 4[(1/2)(n/7) -2]$
  - ⇨ $n - (2n/7 -8)$
  - ⇨ $5n/7 + 8$ →2

- Overall the running time T(n) is determined by combining the above 2 sub problems 1&2 along with O(n), which is work done apart from 2 recursions,
  - ⇨ $T(n) \leq T(n/7) + T(5n/7 + 8) + O(n)$ →3
-
- The above recurrence equation 3 is solved by guessing that -
  - ⇨ $T(n) \leq cn$, for some constant $c > 0$ →4
  - ⇨ $T(n) \leq cn/7 + 5cn/7 + 8c + bn$, where $b>0$ is constant →5

- Solving above 2 equations, from 4 & 5 we get,
  - ⇨ $6cn/7 + 8c + bn \leq cn$ →6
  - ⇨ $bn + 8c \leq cn-(6cn/7)$
  - ⇨ $bn + 8c \leq cn/7$
  - ⇨ $cn/7 - 8c \geq bn$

⇨ c[(n/7)-8] ≥ bn
⇨ c ≥ 7bn/[n-56]

- We should choose n>56 and we should get constant c >0. If we choose n = 2*56 = 112, we get **c ≥ 14b**, which satisfies c>0 for our guess T(n) ≤ cn.
- Also, since the coefficients sum to 1/7 + 5/7 = 6/7 = 0.85 < 1, recursion solves in linear time. Hence the DSelect algorithm with 7 grouping can be solved in linear time **O(n).**

Analysis for Groups of Nine

- The algorithm has 2 recursive calls, the first one is performed on a set of baby medians of size n/5,
  ⇨ g=n/9 →1
- The second recursive call is made either on set L or G based on pivot value X. Consider an example, where there is sequence of 9 distinct elements (N=9). After determining the median (4^{rd} element in sequence S) there would be at least **5** elements <= median i.e 5≤|L|+|E|. We have that for g/2 => **(1/2)(n/9) groups**, at least half of the group elements are less than or equal to X.

- We conclude the second recursive call is performed in set of at most,
  ⇨ n – 5[(1/2)(n/9) -2]
  ⇨ n – (5n/18 -10)
  ⇨ 13n/18 + 10 →2

- Overall the running time T(n) is determined by combining the above 2 sub problems 1&2 along with O(n), which is work done apart from 2 recursions,
  ⇨ T (n) ≤ T (n/9) + T (13n/18 + 10) + O(n) →3
-
- The above recurrence equation 3 is solved by guessing that -
  ⇨ T(n) ≤ cn, for some constant c > 0 →4
  ⇨ T(n) ≤ cn/9 + 13cn/18 +10c+ bn, where b>0 is constant →5

- Solving above 2 equations, from 4 & 5 we get,
  ⇨ 15cn/18 +10c +bn ≤ cn →6
  ⇨ bn + 10c ≤ cn-(15cn/18)
  ⇨ bn + 10c ≤ cn/6
  ⇨ cn/6 – 10c ≥ bn
  ⇨ c[(n/6)-10] ≥ bn
  ⇨ c ≥ 6bn/[n-60]

- We should choose n>60 and we should get constant c >0. If we choose n = 2*60 = 120, we get **c ≥ 12b**, which satisfies c>0 for our guess T(n) ≤ cn.
- Also, since the coefficients sum to 1/9 + 13/18 = 5/6 = 0.83 < 1, recursion solves in linear time. Hence the DSelect algorithm with 9 grouping can be solved in linear time **O(n).**

Best asymptotic characterization

- From the above analysis, we understand that Dselect group 3 has O(nlogn) and is not the best.
- Among the rest of DSelect algorithms, **group 9 has the best asymptotic characterization** as the remaining sub problem size in this case is very small – its coefficients sum to 1/9 + 13/18 = 5/6 = **0.83** < 1, compared to group 7 coefficients sum 1/7 + 5/7 = 6/7 = 0.85 and group 5 coefficients sum 1/5 + 7/10 = 9/10 = 0.9.
- Also for constant size n=120, for group 9 we get, $c \geq 6bn/[n-60]$, which is $c \geq 12b$ which is lesser than group 7 - $c \geq 13.11b$ and group 5 - $c \geq 20b$, for any constant $b>0$.

EXPERIMENTAL RESULTS

Experiments are performed on QuickSelect and Dselect algorithms (groups 3,5,7,9) with 20 different input data size N from 10 to 10Lakh integer values on constant K value.

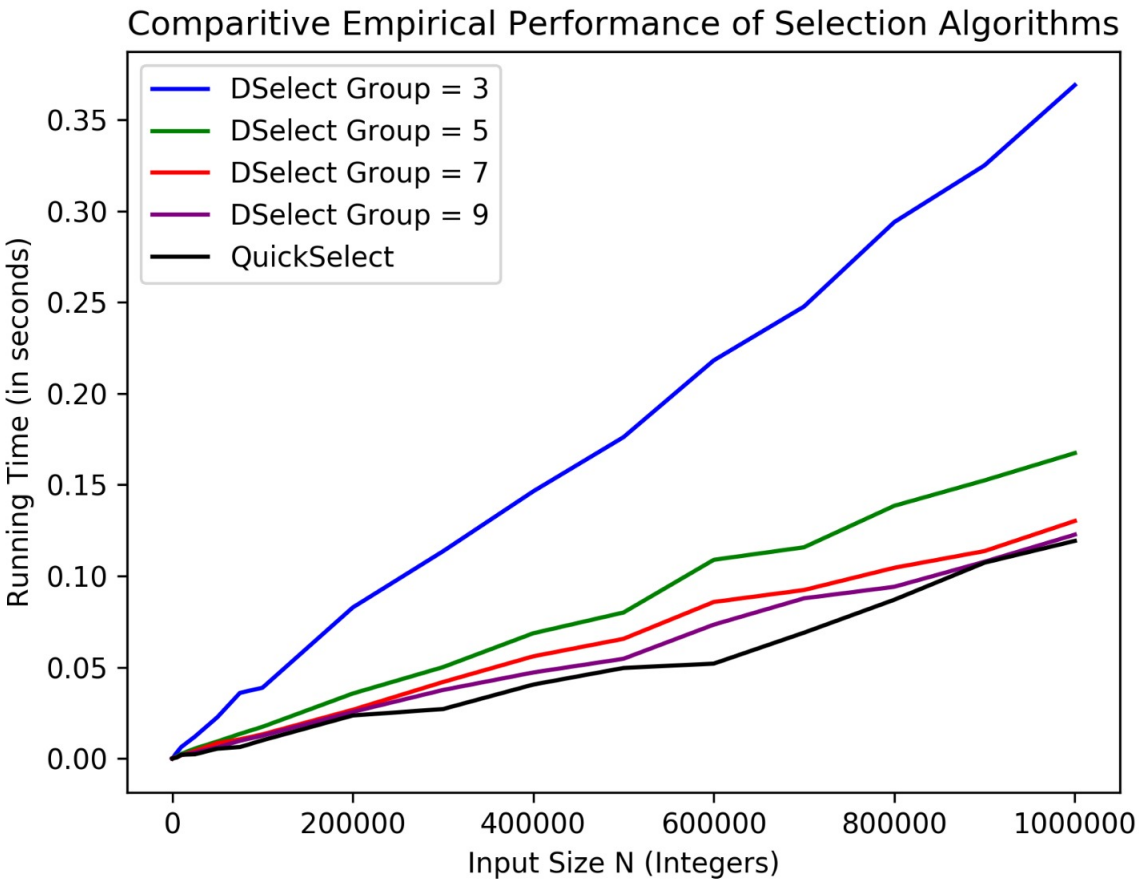
	Input Data Size N	Time of DSelect G=3	Time of DSelect G=5	Time of DSelect G=7	Time of DSelect G=9	Time of Quick Select
2	10	3.00E-05	1.30E-05	1.10E-05	9.00E-06	3.30E-05
3	50	3.60E-05	5.00E-05	4.00E-05	1.50E-05	5.20E-05
4	100	0.000173	0.000104	8.30E-05	8.80E-05	7.10E-05
5	500	0.000414	0.000186	0.000119	0.000118	0.000144
6	1000	0.000582	0.000358	0.000297	0.000206	0.000128
7	5000	0.00331	0.001224	0.000837	0.000795	0.000579
8	10000	0.006329	0.002539	0.001797	0.001882	0.002044
9	25000	0.011981	0.005472	0.004122	0.003278	0.002323
10	50000	0.022677	0.009369	0.008474	0.006116	0.00543
11	75000	0.035987	0.013521	0.010535	0.009735	0.006293
12	100000	0.038816	0.017397	0.013253	0.012525	0.010014
13	200000	0.082779	0.035633	0.026716	0.025556	0.023586
14	300000	0.113585	0.050095	0.041934	0.037568	0.02712
15	400000	0.146433	0.068653	0.056012	0.047168	0.040541
16	500000	0.176121	0.080029	0.065612	0.054678	0.049634
17	600000	0.218192	0.108898	0.085792	0.073368	0.052025
18	700000	0.247698	0.115772	0.092351	0.0878	0.068978
19	800000	0.293951	0.138494	0.104533	0.094121	0.087022
20	900000	0.325052	0.152434	0.113686	0.107848	0.107329
21	1000000	0.369018	0.167388	0.130187	0.122692	0.119266

Measuring elapsed time

- The elapsed time is measured by using suggested Timer.h in C++.
- The timer is started using t.start() before calling the selection algorithm and elapsed time t.elapsedUserTime(a) is calculated soon after the select algorithm call is done.
- The double variable a has the running time of that algorithm.

Comparative results of 5 algorithms

- From the graph, its to be noted that DSelect group 3 takes the longest time. It matches with the theoretical analysis that its running time is O(nlogn).
- The DSelect group 5 takes 2nd longest time, though it runs linear time O(n), followed by group 7 and group 9 takes the least time to run among all DSelect algorithms from the graph.



- This was proved in the theoretical analysis, as the coefficient sums of DSelect algorithms were in order - group 3>group 5>group 7>group 9 i.e 1/3 + 2/3 = **1** > 1/5 + 7/10 = 9/10 = **0.9** > 1/7 + 5/7 = 6/7 = **0.85** > 1/9 + 13/18 = 5/6 = **0.83.**
- Group 9 with least coefficient sum proves to be most efficient among DSelect algorithms from graph.
- **Quick select algorithm performance is best** of all. It is better than DSelect group 9 as we can see from the graph. The quick select algorithm randomizes the pivot selection and worst case never occurs. It works really well for N from 10 to 10Lakh range of integer values.