Quicksort and Selection

Abhiram Ranade

February 12, 2018

Randomized Quicksort

Randomized Quicksort

Quicksort(X){

Randomized Quicksort

Quicksort(X){

1. Pick pivot p at random from X.

Randomized Quicksort

Quicksort(X){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p.   M : equal to p.   L: larger than p.

Randomized Quicksort

Quicksort(X){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p.   M : equal to p.   L: larger than p.
3. Return Quicksort(S) || M || Quicksort(L)

}

Randomized Quicksort

Quicksort(X){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p.   M : equal to p.   L: larger than p.
3. Return Quicksort(S) || M || Quicksort(L)

}

We will estimate expected time, expectation over pivot choices.

Randomized Quicksort

Quicksort(X){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p.   M : equal to p.   L: larger than p.
3. Return Quicksort(S) || M || Quicksort(L)

}

We will estimate expected time, expectation over pivot choices.

Possible to write recurrence.

Randomized Quicksort

Quicksort(X){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.
3. Return Quicksort(S) || M || Quicksort(L)

}

We will estimate expected time, expectation over pivot choices.

Possible to write recurrence. Instead we will use a more global method.

Randomized Quicksort

Quicksort(X){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p.   M : equal to p.   L: larger than p.
3. Return Quicksort(S) || M || Quicksort(L)

}

We will estimate expected time, expectation over pivot choices.

Possible to write recurrence. Instead we will use a more global method.

we show that over the entire execution the expected number of comparisons is $O(n \log n)$, where n is the number of keys.

Randomized Quicksort

Quicksort(X){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p.   M : equal to p.   L: larger than p.
3. Return Quicksort(S) || M || Quicksort(L)

}

We will estimate expected time, expectation over pivot choices.

Possible to write recurrence. Instead we will use a more global method.

we show that over the entire execution the expected number of comparisons is $O(n \log n)$, where n is the number of keys.

Time = O(number of comparisons) $\Rightarrow$ Time $= O(n \log n)$

Analysis

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if ith or jth is picked before elements with rank $i+1, \ldots, j-1$ are picked.

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if ith or jth is picked before elements with rank $i + 1, \ldots, j - 1$ are picked.

all ranks $i, \ldots, j$ equally likely to be picked before the others

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if ith or jth is picked before elements with rank $i+1, \ldots, j-1$ are picked.

all ranks $i, \ldots, j$ equally likely to be picked before the others

Probability: $2/(j-i+1)$

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if ith or jth is picked before elements with rank $i+1, \ldots, j-1$ are picked.

all ranks $i, \ldots, j$ equally likely to be picked before the others

Probability: $2/(j-i+1)$

$$\sum_{i<j} E[x_{ij}] \quad =$$

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if ith or jth is picked before elements with rank $i+1, \ldots, j-1$ are picked.

all ranks $i, \ldots, j$ equally likely to be picked before the others

Probability: $2/(j-i+1)$

$$\sum_{i<j} E[x_{ij}] = \sum_{j=2}^{j=n} \sum_{i=1}^{i=j-1} \frac{2}{j-i+1}$$

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if ith or jth is picked before elements with rank $i+1, \ldots, j-1$ are picked.

all ranks $i, \ldots, j$ equally likely to be picked before the others

Probability: $2/(j-i+1)$

$$\sum_{i<j} E[x_{ij}] = \sum_{j=2}^{j=n} \sum_{i=1}^{i=j-1} \frac{2}{j-i+1} = \sum_{j=2}^{j=n} \frac{2}{j} + \ldots + \frac{2}{2}$$

Analysis

Random variable x_{ij} = whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if ith or jth is picked before elements with rank $i+1, \ldots, j-1$ are picked.

all ranks $i, \ldots, j$ equally likely to be picked before the others

Probability: $2/(j-i+1)$

$$\begin{aligned}\sum_{i<j} E[x_{ij}] &= \sum_{j=2}^{j=n} \sum_{i=1}^{i=j-1} \frac{2}{j-i+1} = \sum_{j=2}^{j=n} \frac{2}{j} + \ldots + \frac{2}{2} \\ &\leq \sum_{j=2}^{j=n} 2 \ln j\end{aligned}$$

Analysis

Random variable $x_{ij} =$ whether ith smallest key is compared with the jth smallest.

Doesnt matter where they are in the given list.

Expected number of comparisons: $\sum_{i<j} x_{ij}$

x_{ij} is a 0-1 variable $\Rightarrow E[x_{ij}] = p(x_{ij} = 1)$

ith smallest, jth smallest compared if ith or jth is picked before elements with rank $i+1, \ldots, j-1$ are picked.

all ranks $i, \ldots, j$ equally likely to be picked before the others

Probability: $2/(j - i + 1)$

$$
\begin{aligned}
\sum_{i<j} E[x_{ij}] &= \sum_{j=2}^{j=n} \sum_{i=1}^{i=j-1} \frac{2}{j-i+1} = \sum_{j=2}^{j=n} \frac{2}{j} + \ldots + \frac{2}{2} \\
&\leq \sum_{j=2}^{j=n} 2 \ln j \leq 2n \ln n
\end{aligned}
$$

Remarks

Remarks

- In actual implementations, quicksort is done "in-place": S list is built from the beginning of the array, and L from the end.

Remarks

- In actual implementations, quicksort is done "in-place": S list is built from the beginning of the array, and L from the end.
- Quicksort is among the fastest sorting algorithms.

Remarks

- In actual implementations, quicksort is done "in-place": S list is built from the beginning of the array, and L from the end.
- Quicksort is among the fastest sorting algorithms.
- Make sure you understand what we proved: Expected time for any instance is $O(n \log n)$. Hence also the expected time for the worst instance.

Selection

Selection

Input: Keys $x_1, \ldots, x_n$, Integer r.

Selection

Input: Keys $x_1, \ldots, x_n$, Integer r.
Output: rth smallest key.

Selection

Input: Keys $x_1, \ldots, x_n$, Integer r.
Output: rth smallest key.

"Easy" algorithm: Sort x_i. Return rth from resulting order.

Selection

Input: Keys $x_1, \ldots, x_n$, Integer r.
Output: rth smallest key.

"Easy" algorithm: Sort x_i. Return rth from resulting order.
Time: $O(n \log n)$

Selection

Input: Keys $x_1, \ldots, x_n$, Integer r.
Output: rth smallest key.

"Easy" algorithm: Sort x_i. Return rth from resulting order.
Time: $O(n \log n)$

Can be done in $O(n)$ time. Next.

Idea: Adapt Quicksort

Idea: Adapt Quicksort

Quickselect(X,r){

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.
3. If $r \leq |S|$ return Quickselect(S,r).

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.
3. If $r \leq |S|$ return Quickselect(S,r).
   Else if $r \leq |S| + |M|$ return p.

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.
3. If $r \leq |S|$ return Quickselect(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Quickselect(L, $r - |S| - |M|$).

}

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p.   M : equal to p.   L: larger than p.
3. If $r \leq |S|$ return Quickselect(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Quickselect(L, $r - |S| - |M|$).

}

Adapt Quicksort analysis to give $O(n)$ time   Homework

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.
3. If $r \leq |S|$ return Quickselect(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Quickselect(L, $r - |S| - |M|$).

}

Adapt Quicksort analysis to give $O(n)$ time Homework

Today:

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.
3. If $r \leq |S|$ return Quickselect(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Quickselect(L, $r - |S| - |M|$).

}

Adapt Quicksort analysis to give $O(n)$ time Homework

Today:
Deterministic way to ensure p = approximate median.

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.
3. If $r \leq |S|$ return Quickselect(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Quickselect(L, $r - |S| - |M|$).

}

Adapt Quicksort analysis to give $O(n)$ time Homework

Today:
Deterministic way to ensure p = approximate median.
$\Rightarrow$ Problem nearly halves in each recursion.

Idea: Adapt Quicksort

Quickselect(X,r){

1. Pick pivot p at random from X.
2. Distribute elements of X into 3 lists:
   S : smaller than p. M : equal to p. L: larger than p.
3. If $r \leq |S|$ return Quickselect(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Quickselect(L, $r - |S| - |M|$).

}

Adapt Quicksort analysis to give $O(n)$ time Homework

Today:
Deterministic way to ensure p = approximate median.
$\Rightarrow$ Problem nearly halves in each recursion.
$\Rightarrow$ Total time $= O(n)$.

Deterministic selection algorithm

Deterministic selection algorithm

Select (X,r){

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!
3. Distribute X into lists:

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!
3. Distribute X into lists:
   S : smaller than p, M: equal to p, L: larger than p.

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!
3. Distribute X into lists:
   S : smaller than p, M: equal to p, L: larger than p.
4. If $r \leq |S|$ return Select(S,r).

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!
3. Distribute X into lists:
   S : smaller than p, M: equal to p, L: larger than p.
4. If $r \leq |S|$ return Select(S,r).
   Else if $r \leq |S| + |M|$ return p.

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!
3. Distribute X into lists:
   S : smaller than p, M: equal to p, L: larger than p.
4. If $r \leq |S|$ return Select(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Select$(L, r - |S| - |M|)$.

}

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!
3. Distribute X into lists:
   S : smaller than p, M: equal to p, L: larger than p.
4. If $r \leq |S|$ return Select(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Select$(L, r - |S| - |M|)$.

}

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$ Proof soon.

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!
3. Distribute X into lists:
   S : smaller than p, M: equal to p, L: larger than p.
4. If $r \leq |S|$ return Select(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Select$(L, r - |S| - |M|)$.

}

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$ Proof soon.

Lemma: $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$ Obvious.

Deterministic selection algorithm

Select (X,r){

1. If $n = |X| < 50$, sort and return rth.
2. Choose pivot p as follows
   - Create $\lfloor n/5 \rfloor$ 5 tuples from X. Some elements remain.
   - Find m_i = median of ith 5-tuple.
   - p = median of all m_i. Recurse!
3. Distribute X into lists:
   S : smaller than p, M: equal to p, L: larger than p.
4. If $r \leq |S|$ return Select(S,r).
   Else if $r \leq |S| + |M|$ return p.
   Else return Select$(L, r - |S| - |M|)$.

}

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$ Proof soon.

Lemma: $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$ Obvious.

Lemma: Using recursion tree $T(n) = O(n)$ Proof soon.

Lemma: $|L|, |S| \le n - 3\lfloor n/10 \rfloor$

$p =$ median of $\lfloor n/5 \rfloor$ medians of 5 tuples.

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$

$p =$ median of $\lfloor n/5 \rfloor$ medians of 5 tuples.

Number of medians less than or equal to p : $\lfloor \lfloor n/5 \rfloor /2 \rfloor$

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$

$p =$ median of $\lfloor n/5 \rfloor$ medians of 5 tuples.

Number of medians less than or equal to p : $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$

$p =$ median of $\lfloor n/5 \rfloor$ medians of 5 tuples.

Number of medians less than or equal to p : $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$

Proof of equality: Let $n = 10p + q \ldots$

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$

$p =$ median of $\lfloor n/5 \rfloor$ medians of 5 tuples.

Number of medians less than or equal to p : $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$

Proof of equality: Let $n = 10p + q \ldots$

Each median is less than or equal to 3 elements in its 5 tuple.

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$

$p =$ median of $\lfloor n/5 \rfloor$ medians of 5 tuples.

Number of medians less than or equal to p : $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$

Proof of equality: Let $n = 10p + q \ldots$

Each median is less than or equal to 3 elements in its 5 tuple.

Number of elements less than or equal to p : $\geq 3\lfloor n/10 \rfloor$.

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$

$p =$ median of $\lfloor n/5 \rfloor$ medians of 5 tuples.

Number of medians less than or equal to p : $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$

Proof of equality: Let $n = 10p + q \ldots$

Each median is less than or equal to 3 elements in its 5 tuple.

Number of elements less than or equal to p : $\geq 3\lfloor n/10 \rfloor$.

Thus $|L| < n - 3\lfloor n/10 \rfloor$.

Lemma: $|L|, |S| \leq n - 3\lfloor n/10 \rfloor$

$p =$ median of $\lfloor n/5 \rfloor$ medians of 5 tuples.

Number of medians less than or equal to p : $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$

Proof of equality: Let $n = 10p + q \ldots$

Each median is less than or equal to 3 elements in its 5 tuple.

Number of elements less than or equal to p : $\geq 3\lfloor n/10 \rfloor$.

Thus $|L| < n - 3\lfloor n/10 \rfloor$.

Similarly $|S|$.

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.

$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work $=$ O(work at top level)

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work $=$ O(work at top level) $= O(n)$.

Analysis accounting for floor:

Solving $T(n) \le T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.

$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.

Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Analysis accounting for floor:

We stop recursion when $n < 50 \quad \Rightarrow \quad$ In the recurrence $n \ge 50$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50 \quad \Rightarrow \quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq$

Solving $T(n) \le T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\quad\Rightarrow\quad$ In the recurrence $n \ge 50$

Thus $\lfloor n/10 \rfloor \ge n/10 - 1$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work $=$ O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\quad\Rightarrow\quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work $=$ O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\quad\Rightarrow\quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50 \quad \Rightarrow \quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \;+\; (n - 3\lfloor n/10 \rfloor$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work $=$ O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50 \quad \Rightarrow \quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \; + \; (n - 3\lfloor n/10 \rfloor$
$\leq \frac{n}{50}(10$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50 \quad \Rightarrow \quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \ + \ (n - 3\lfloor n/10 \rfloor$
$\leq \frac{n}{50}(10 + 50$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) = $O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\quad \Rightarrow \quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \; + \; (n - 3\lfloor n/10 \rfloor$
$\leq \frac{n}{50}(10 + 50 - 3 \times 4)$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\quad \Rightarrow \quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \; + \; (n - 3\lfloor n/10 \rfloor$
$\leq \frac{n}{50}(10 + 50 - 3 \times 4) = n\frac{24}{25}$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50 \quad \Rightarrow \quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \; + \; (n - 3\lfloor n/10 \rfloor$
$\leq \frac{n}{50}(10 + 50 - 3 \times 4) = n\frac{24}{25}$

Thus constant fraction reduction even with floors!

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work $=$ O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\quad \Rightarrow \quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \ + \ (n - 3\lfloor n/10 \rfloor$
$\leq \frac{n}{50}(10 + 50 - 3 \times 4) = n\frac{24}{25}$

Thus constant fraction reduction even with floors!

Time for leaf level:

Solving $T(n) \le T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) = $O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\Rightarrow$ In the recurrence $n \ge 50$

Thus $\lfloor n/10 \rfloor \ge n/10 - 1 \ge n/10 - n/50 \ge 4n/50$.

Thus subproblem size sum $\le n/5 + (n - 3\lfloor n/10 \rfloor$
$\le \frac{n}{50}(10 + 50 - 3 \times 4) = n\frac{24}{25}$
Thus constant fraction reduction even with floors!

Time for leaf level: Time for 50 element sequence $= c'$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work = O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\quad\Rightarrow\quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \; + \; (n - 3\lfloor n/10 \rfloor$
$\leq \frac{n}{50}(10 + 50 - 3 \times 4) = n\frac{24}{25}$

Thus constant fraction reduction even with floors!

Time for leaf level: Time for 50 element sequence $= c'$
Total leaf level time $\leq c'n$

Solving $T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + cn$

Ignoring floors: Subproblem sizes add up to $9n/10$.
$\Rightarrow$ Work reduces by factor $9/10$ each level of recursion tree.
Work decreases geometrically by levels, so work $=$ O(work at top level) $= O(n)$.

Analysis accounting for floor:
We stop recursion when $n < 50$ $\quad\Rightarrow\quad$ In the recurrence $n \geq 50$

Thus $\lfloor n/10 \rfloor \geq n/10 - 1 \geq n/10 - n/50 \geq 4n/50$.

Thus subproblem size sum $\leq n/5 \; + \; (n - 3\lfloor n/10 \rfloor$
$\leq \frac{n}{50}(10 + 50 - 3 \times 4) = n\frac{24}{25}$
Thus constant fraction reduction even with floors!

Time for leaf level: Time for 50 element sequence $= c'$
Total leaf level time $\leq c'n$

Overall: $O(n)$

Remarks

Remarks

- Clever recursive use of selection itself to find pivot.

Remarks

- Clever recursive use of selection itself to find pivot.
- Not quite "divide" and conquer: Two recursive calls overlap in elements.

Remarks

- Clever recursive use of selection itself to find pivot.
- Not quite "divide" and conquer: Two recursive calls overlap in elements.
- Need to be careful to ensure that the two subproblems have total size $\leq n$.

Remarks

- Clever recursive use of selection itself to find pivot.
- Not quite "divide" and conquer: Two recursive calls overlap in elements.
- Need to be careful to ensure that the two subproblems have total size $\leq n$.
- Exercise: Will the algorithm work if we took medians of 3-tuples or of 7-tuples?

Remarks

- Clever recursive use of selection itself to find pivot.
- Not quite "divide" and conquer: Two recursive calls overlap in elements.
- Need to be careful to ensure that the two subproblems have total size $\leq n$.
- Exercise: Will the algorithm work if we took medians of 3-tuples or of 7-tuples?
- We can improve constants, e.g. 24/25. Key emphasis in this course: do not worry about the constant, but get a simple clean argument.

A detour: some more notation

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f \,|\, \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f \mid \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

$\omega(g) = \{f \mid \lim_{n\to\infty} f(n)/g(n) = \infty\}$ $\quad$ $f = o(g)$ means "$f > g$"

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

$\omega(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = \infty\}$ $\quad$ $f = o(g)$ means "$f > g$"

Example: $n = o(n \log n)$. $n^3 = \omega(n^2)$.

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f \mid \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

$\omega(g) = \{f \mid \lim_{n\to\infty} f(n)/g(n) = \infty\}$ $\quad$ $f = o(g)$ means "$f > g$"

Example: $n = o(n \log n)$. $n^3 = \omega(n^2)$.

"Is there an $o(n \log n)$ time sorting algorithm?"

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

$\omega(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = \infty\}$ $\quad$ $f = o(g)$ means "$f > g$"

Example: $n = o(n \log n)$. $n^3 = \omega(n^2)$.

"Is there an $o(n \log n)$ time sorting algorithm?"

Limits and asymptotic comparisons:

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

$\omega(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = \infty\}$ $\quad$ $f = o(g)$ means "$f > g$"

Example: $n = o(n \log n)$. $n^3 = \omega(n^2)$.

"Is there an $o(n \log n)$ time sorting algorithm?"

Limits and asymptotic comparisons:

- $\lim_{n\to\infty} f(n)/g(n) = 0 : f = o(g)$

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

$\omega(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = \infty\}$ $\quad$ $f = o(g)$ means "$f > g$"

Example: $n = o(n \log n)$. $n^3 = \omega(n^2)$.

"Is there an $o(n \log n)$ time sorting algorithm?"

Limits and asymptotic comparisons:

- $\lim_{n\to\infty} f(n)/g(n) = 0$: $f = o(g)$
- $\lim_{n\to\infty} f(n)/g(n)$ is finite and positive: $f = \theta(g)$

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

$\omega(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = \infty\}$ $\quad$ $f = o(g)$ means "$f > g$"

Example: $n = o(n \log n)$. $n^3 = \omega(n^2)$.

"Is there an $o(n \log n)$ time sorting algorithm?"

Limits and asymptotic comparisons:

- $\lim_{n\to\infty} f(n)/g(n) = 0$: $f = o(g)$
- $\lim_{n\to\infty} f(n)/g(n)$ is finite and positive: $f = \theta(g)$
- $\lim_{n\to\infty} f(n)/g(n) = \infty$: $f = \omega(g)$

A detour: some more notation

We said "θ means $=$", "O means $\leq$", "Ω means $\geq$"

Asymptotically, and ignoring constants

Classes Little o, Little omega: "mean $<, >$"

$o(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = 0\}$ $\quad$ $f = o(g)$ means "$f < g$"

$\omega(g) = \{f | \lim_{n\to\infty} f(n)/g(n) = \infty\}$ $\quad$ $f = o(g)$ means "$f > g$"

Example: $n = o(n \log n)$. $n^3 = \omega(n^2)$.

"Is there an $o(n \log n)$ time sorting algorithm?"

Limits and asymptotic comparisons:

- $\lim_{n\to\infty} f(n)/g(n) = 0$: $f = o(g)$
- $\lim_{n\to\infty} f(n)/g(n)$ is finite and positive: $f = \theta(g)$
- $\lim_{n\to\infty} f(n)/g(n) = \infty$: $f = \omega(g)$

$o(g) \subset O(g), \quad \omega(g) \subset \Omega(g)$