CSE 312

Foundations of Computing II

Randomized Algorithms

Outline

1 Minimum

2 Quick Sort

Minimum Algorithm


```
1 // precondition: A is non-empty
2 def min(A, N):
3    result = ∞
4    for t = 1 to N:
5       if A[t] < result:
6          result = A[t]
```

We've already seen an algorithm like this one on the homework where we analyzed time to the first execution of line 6.

This time, let's analyze **how many times** line 6 gets executed.

Worst Case

Minimum Algorithm


```
1 // precondition: A is non-empty
2 def min(A, N):
3    result = ∞
4    for t = 1 to N:
5       if A[t] < result:
6          result = A[t]
```

We've already seen an algorithm like this one on the homework where we analyzed time to the first execution of line 6.

This time, let's analyze **how many times** line 6 gets executed.

Worst Case

The worst case is reverse-sorted order which is $\mathcal{O}(N)$.

Average Case?

What does this even mean?

Minimum Algorithm

```
1 // precondition: A is non-empty
2 def min(A, N):
3    result = ∞
4    for t = 1 to N:
5       if A[t] < result:
6          result = A[t]
```

Average Case Assumptions

- Each permutation of the input is equally likely
- No equal entries

Average Case Analysis

Let X be a r.v. for the number of times line 6 is executed.

Minimum Algorithm


```
1 // precondition: A is non-empty
2 def min(A, N):
3    result = ∞
4    for t = 1 to N:
5       if A[t] < result:
6          result = A[t]
```

Average Case Assumptions

- Each permutation of the input is equally likely
- No equal entries

Average Case Analysis

Let X be a r.v. for the number of times line 6 is executed. Define X_i as an i.r.v. for "whether or not the ith index results in an executation of line 6. Then, note that $X = \sum X_i$. So, $\mathbb{E}[X] = \sum \mathbb{E}[X_i] = \sum \Pr(X_i = 1)$. Note that $X_i = 1$ exactly when $\texttt{A}[i]$ is the minimum in $\{\texttt{A}[1], \ldots, \texttt{A}[i]\}$. Let T be the set of t-subsets of elements of A.

Minimum Algorithm

```
1 // precondition: A is non−empty
2 def min(A, N):
3    result = ∞
4    for t = 1 to N:
5       if A[t] < result:
6          result = A[t]
```

Average Case Analysis

Let X be a r.v. for the number of times line 6 is executed. Define X_i as an i.r.v. for "whether or not the ith index results in an executation of line 6. Then, note that $X = \sum X_i$. So, $\mathbb{E}[X] = \sum \mathbb{E}[X_i] = \sum \Pr(X_i = 1)$.
Note that $X_i = 1$ exactly when $\texttt{A}[i]$ is the minimum in $\{\texttt{A}[1], \ldots, \texttt{A}[i]\}$.
Let T be the set of t-subsets of elements of A. Then:

Minimum Algorithm

```
1 // precondition: A is non-empty
2 def min(A, N):
3    result = ∞
4    for t = 1 to N:
5       if A[t] < result:
6          result = A[t]
```

Average Case Analysis

Let X be a r.v. for the number of times line 6 is executed. Define X_i as an i.r.v. for "whether or not the ith index results in an executation of line 6. Then, note that $X = \sum X_i$. So, $\mathbb{E}[X] = \sum \mathbb{E}[X_i] = \sum \Pr(X_i = 1)$.
Note that $X_i = 1$ exactly when $\texttt{A}[i]$ is the minimum in $\{\texttt{A}[1], \ldots, \texttt{A}[i]\}$.
Let T be the set of t-subsets of elements of A. Then:

$$\Pr(X_t = 1) = \sum_{x \in T} \Pr(X_t = 1 \mid x) \Pr(x)$$

Minimum Algorithm


```
1 // precondition: A is non-empty
2 def min(A, N):
3    result = ∞
4    for t = 1 to N:
5       if A[t] < result:
6          result = A[t]
```

Average Case Analysis

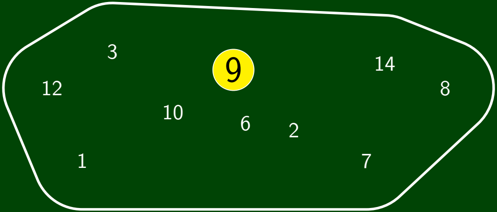
Let X be a r.v. for the number of times line 6 is executed. Define X_i as an i.r.v. for "whether or not the ith index results in an executation of line 6. Then, note that $X = \sum X_i$. So, $\mathbb{E}[X] = \sum \mathbb{E}[X_i] = \sum \Pr(X_i = 1)$. Note that $X_i = 1$ exactly when $\texttt{A}[i]$ is the minimum in $\{\texttt{A}[1], \ldots, \texttt{A}[i]\}$. Let T be the set of t-subsets of elements of A. Then:

$$\begin{aligned}
\Pr(X_t = 1) &= \sum_{x \in T} \Pr(X_t = 1 \mid x) \Pr(x) \\
&= \sum_{x \in T} \frac{(t-1)!}{t!} \frac{1}{\binom{N}{t}} \\
&= \frac{(t-1)!}{t!} = \frac{1}{t}
\end{aligned}$$

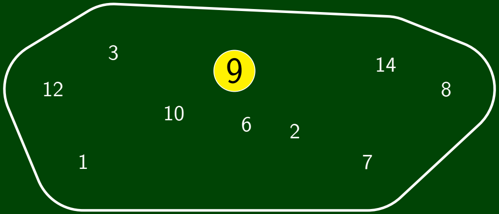
Outline

1 Minimum

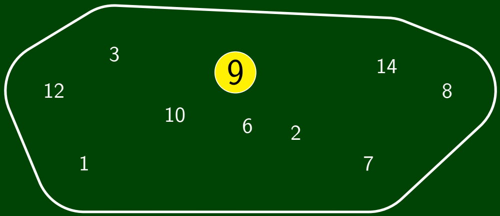
2 Quick Sort


3
9
14
12
8
10
6
2
1
7

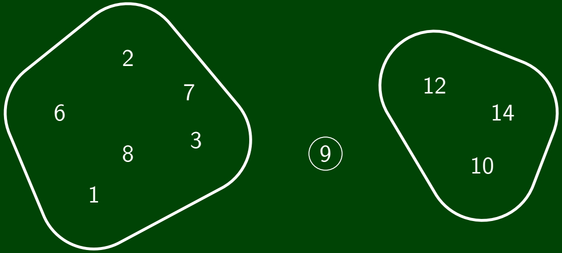
Quick Sort

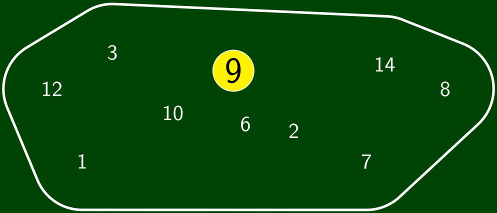


Partition based on `pivot = 9`

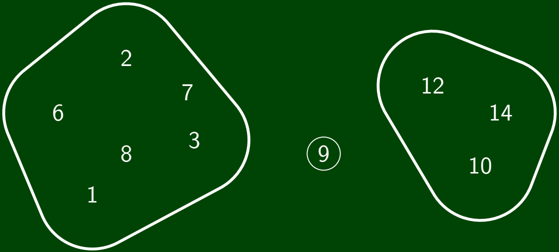


Partition based on `pivot = 9`

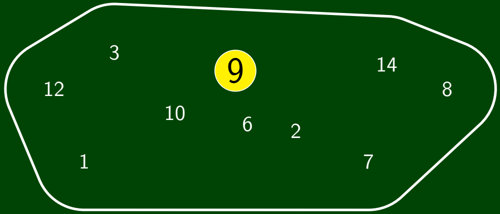




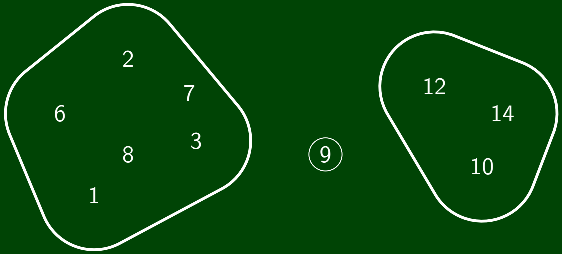
Partition based on `pivot = 9`



Recursively Sort Halves



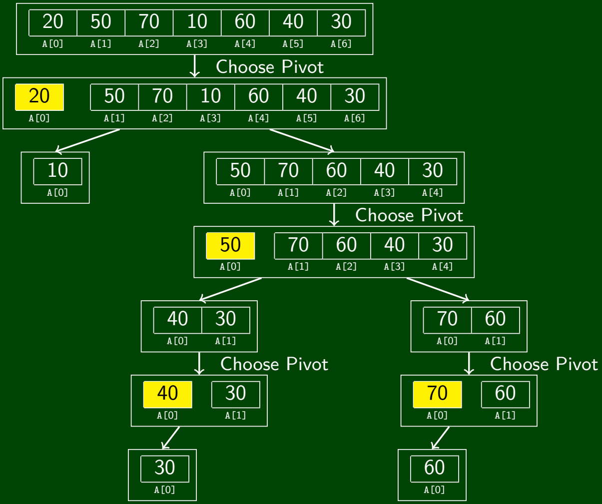
Partition based on `pivot = 9`



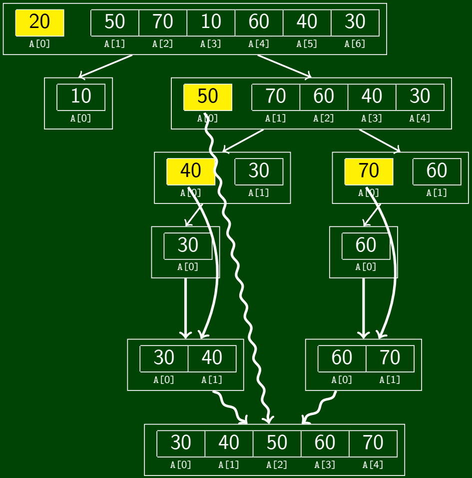
Recursively Sort Halves

1	2	3	6	7	8	9	10	12	14
L[0]	L[1]	L[2]	L[3]	L[4]	L[5]		R[0]	R[1]	R[2]

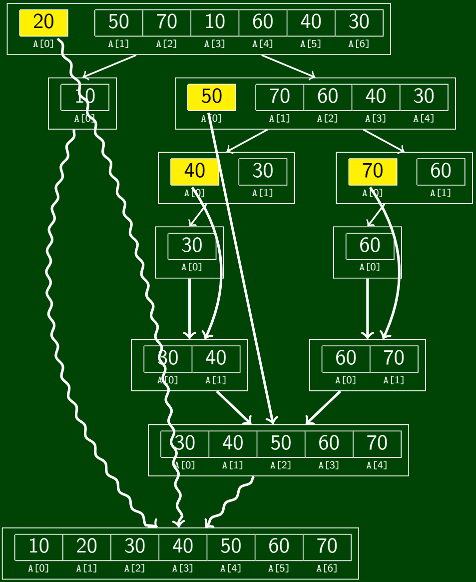
Quick Sort



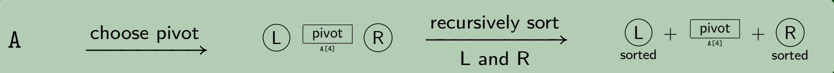
20 50 70 10 60 40 30
A[0] A[1] A[2] A[3] A[4] A[5] A[6]
Choose Pivot
20
A[0]
50 70 10 60 40 30
A[1] A[2] A[3] A[4] A[5] A[6]
10
A[0]
50 70 60 40 30
A[0] A[1] A[2] A[3] A[4]
Choose Pivot
50
A[0]
70 60 40 30
A[1] A[2] A[3] A[4]
40 30
A[0] A[1]
70 60
A[0] A[1]
Choose Pivot
Choose Pivot
40
A[0]
30
A[1]
70
A[0]
60
A[1]
30
A[0]
60
A[0]


20 A[0]
50 70 10 60 40 30
A[1] A[2] A[3] A[4] A[5] A[6]
10 A[0]
50 A[0]
70 60 40 30
A[1] A[2] A[3] A[4]
40 A[0] 30 A[1]
70 A[0] 60 A[1]
30 A[0]
60 A[0]
30 40 A[0] A[1]
60 70 A[0] A[1]
30 40 50 60 70
A[0] A[1] A[2] A[3] A[4]

Quick Sort



20 50 70 10 60 40 30
A[0] A[1] A[2] A[3] A[4] A[5] A[6]
10
A[0]
50 70 60 40 30
A[0] A[1] A[2] A[3] A[4]
40 30
A[0] A[1]
70 60
A[0] A[1]
30
A[0]
60
A[0]
30 40
A[0] A[1]
60 70
A[0] A[1]
30 40 50 60 70
A[0] A[1] A[2] A[3] A[4]
10 20 30 40 50 60 70
A[0] A[1] A[2] A[3] A[4] A[5] A[6]



Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A,
        pivot));
   return left + pivot + right;
}
```

Runtime and Analysis

- Best Case?
- Worst Case?
- Average Case?

Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A, pivot));
   return left + pivot + right;
}
```

Average Case Assumptions

- Each permutation of the input is equally likely
- No equal entries

Average Case Analysis

Let X be a r.v. for the number of comparisons.

Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A, pivot));
   return left + pivot + right;
}
```

Average Case Assumptions

- Each permutation of the input is equally likely
- No equal entries

Average Case Analysis

Let X be a r.v. for the number of comparisons. Define $X_{i,j}$ as an i.r.v. for "whether or not the ith element in sorted order gets compared with the jth element in sorted order. Then, note that $X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}$.

Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A, pivot));
   return left + pivot + right;
}
```

Average Case Analysis

Then, note that $X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}$.

Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A, pivot));
   return left + pivot + right;
}
```

Average Case Analysis

Then, note that $X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}$. So:

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}\left[X_{i,j}\right] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr\left(X_{i,j} = 1\right)$$

Consider, $X_{i,j}$ for $i < j$. (Note that this is the only case we need to consider because our summation ensures this.)

Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A, pivot));
   return left + pivot + right;
}
```

Average Case Analysis

Consider $X_{i,j}$ for $i < j$. Now, a couple of notes:

Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A, pivot));
   return left + pivot + right;
}
```

Average Case Analysis

Consider $X_{i,j}$ for $i < j$. Now, a couple of notes:

- We will choose a pivot between `A[`i`]` and `A[`j`]` on a recursive call.

Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A, pivot));
   return left + pivot + right;
}
```

Average Case Analysis

Consider $X_{i,j}$ for $i < j$. Now, a couple of notes:

- We will choose a pivot between `A[`i`]` and `A[`j`]` on a recursive call.
- If the choice is strictly between `A[`i`]` and `A[`j`]`, then we will never compare them.

Algorithm

```
quicksort(A) {
   if (A.length < 2) {
      return A;
   }

   pivot = A[RollDie(|A|)]
   left = quicksort(getLess(A, pivot));
   right = quicksort(getGreater(A, pivot));
   return left + pivot + right;
}
```

Average Case Analysis

Consider $X_{i,j}$ for $i < j$. Now, a couple of notes:

- We will choose a pivot between `A[`i`]` and `A[`j`]` on a recursive call.
- If the choice is strictly between `A[`i`]` and `A[`j`]`, then we will never compare them.
- If the choice is equal to `A[`i`]` or `A[`j`]`, then we will always compare them.

Average Case Analysis

Consider $X_{i,j}$ for $i < j$.

- If the choice is strictly between `A[`i`]` and `A[`j`]`, then we will never compare them.
- If the choice is equal to `A[`i`]` or `A[`j`]`, then we will always compare them.
- We will choose a pivot between (or at) `A[`i`]` and `A[`j`]` on some recursive call.

Let $P_{i,j,r}$ be the event "a pivot between i and j is chosen on the rth recursive call.

Average Case Analysis

Consider $X_{i,j}$ for $i < j$.

- If the choice is strictly between `A[`i`]` and `A[`j`]`, then we will never compare them.
- If the choice is equal to `A[`i`]` or `A[`j`]`, then we will always compare them.
- We will choose a pivot between (or at) `A[`i`]` and `A[`j`]` on some recursive call.

Let $P_{i,j,r}$ be the event "a pivot between i and j is chosen on the rth recursive call.

$$\begin{aligned}
\Pr\left(X_{i,j} = 1\right) &= \sum_r \Pr\left(X_{i,j} = 1 \mid P_{i,j,r}\right) \Pr\left(P_{i,j,r}\right) \\
&= \sum_r \frac{2}{j-i+1} \Pr\left(P_{i,j,r}\right) \\
&= \frac{2}{j-i+1} \sum_r \Pr\left(P_{i,j,r}\right) \\
&= \frac{2}{j-i+1}
\end{aligned}$$

Average Case Analysis

Let $P_{i,j,r}$ be the event "a pivot between i and j is chosen on the rth recursive call.

$$\Pr\left(X_{i,j} = 1\right) = \frac{2}{j - i + 1}$$

Average Case Analysis

Let $P_{i,j,r}$ be the event "a pivot between i and j is chosen on the rth recursive call.

$$\Pr\left(X_{i,j} = 1\right) = \frac{2}{j-i+1}$$

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr\left(X_{i,j} = 1\right) \\
&= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} 2\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-i+1}\right) \\
&= \sum_{i=1}^{n} 2(H_n - 1) \\
&< 2n(H_n - 1) \\
&\le 2n\ln(n)
\end{aligned}$$