Thanks to Miles Jones, Russell Impagliazzo, and Sanjoy Dasgupta at UCSD for these slides.

Divide and Conquer sort

- Starting with a list of integers, the goal is to output the list in sorted order.

- Break a problem into similar subproblems
  - Split the list into two sublists each of half the size
- Solve each subproblem recursively
  - recursively sort the two sublists
- Combine
  - put the two sorted sublists together to create a sorted list of all the elements.

MergeSort

```
function mergesort($a[1 \dots n]$)
    if $n > 1$:
        ML = mergesort$\left(a\left[1 \dots \left\lfloor\frac{n}{2}\right\rfloor\right]\right)$
        MR = mergesort $\left(a\left[\left\lfloor\frac{n}{2}\right\rfloor + 1, \dots n\right]\right)$
        return merge(ML,MR)
    else:
        return $a$
```

Median

- The median of a list of numbers is the *middle* number in the list.
- If the list has n values and n is odd, then the middle element is clear. It is the $\lceil n/2 \rceil^{\text{th}}$ smallest element.
- Example:

$$med(8,2,9,11,4) = 8$$

because $n = 5$ and 8 is the $3rd = \lceil 5/2 \rceil^{\text{th}}$ smallest element of the list.

Median

- The median of a list of numbers is the middle number in the list.
- If the list has n values and n is even, then there are two middle elements. Let's say that the median is the $(\frac{n}{2})^{\text{th}}$ smallest element. Then in either case the median is the $\lceil n/2 \rceil^{\text{th}}$ smallest element
- Example:

$$med(10,23,7,26,17,3) = 10$$

because $n = 6$ and 10 is the $3rd = \lceil 6/2 \rceil^{\text{th}}$ smallest element of the list.

Median

- The purpose of the median is to summarize a set of numbers. The *average* is also a commonly used value. The median is more typical of the data.
- For example, suppose in a company with 20 employees, the CEO makes 1 million and all the other workers each make 50,000.
- Then the average is 97,500 and the median is 50,000, which is much closer to the typical worker’s salary.

Median (algorithm)

- Can you think of an efficient way to find the median?
- How long would it take?
- Is there a lower bound on the runtime of all median selection algorithms?

Median (algorithm)

- Can you think of an efficient way to find the median?
- How long would it take?
- Is there a lower bound on the runtime of all median selection algorithms?

- Sort the list then find the $\lceil n/2 \rceil^{\text{th}}$ element $O(n \log n)$.
- You can never have a faster runtime than $O(n)$ because you at least have to look at every element.
- All selection algorithms are $\Omega(n)$

Selection

- What if we designed an algorithm that takes as input, a list of numbers of length n and an integer $1 \leq k \leq n$ and outputs the k^{th} smallest integer in the list.
- Then we could just plug in $\lceil n/2 \rceil$ for k and we could find the median!!

Selection

- Let’s think about selection in a divide and conquer type of way.

- Break a problem into similar subproblems
  - Split the list into two sublists
- Solve each subproblem recursively
  - recursively select from one of the sublists
- Combine

Selection

- How would you split the list?
- Just splitting the list down the middle does not help so much.

- What we will do is pick a random “pivot” and split the list into all integers greater than the pivot and all that are less than the pivot.

- Then we can determine which list to look in to find the k^{th} smallest element. (Note that the value of k may change depending on which list we are looking in.)

Selection

- Example:
- Selection([40,31,6,51,76,58,97,37,86,31,19,30,68],7)

- pick a random pivot….. say 31. Then divide the list into three groups SL, Sv, SR such that SL contains all elements smaller than 31, Sv is all elements equal to 31 and SR is all elements greater than 31.

- SL=[6,19,30], size = 3
- Sv=[31,31], size = 2
- SR=[40,51,76,58,97,37,86,68], size = 8

Selection

- Selection([40,31,6,51,76,58,97,37,86,31,19,30,68],7)

- SL=[6,19,30], size = 3
- Sv=[31,31], size = 2
- SR=[40,51,76,58,97,37,86,68], size = 8

- Now, since k=7 is bigger than the size of SL, we know the kth biggest element cannot be in SL. Since it is bigger than size of SL plus size of Sv, it cannot be in Sv, either. Therefore it must be in SR.

- So the 7th biggest element in the original list is what number in SR?

Selection

- So the 7th biggest element in the original list is the 2nd biggest in SR?

- Selection([40,31,6,51,76,58,97,37,86,31,19,30,68],7)

- SL=[6,19,30], size = 3
- Sv=[31,31], size = 2
- SR=[40,51,76,58,97,37,86,68], size = 8

- Selection([40,31,6,51,76,58,97,37,86,31,19,30,68],7)
  =Selection ([40,51,76,58,97,37,86,68],2)

Selection (Algorithm)

- Input: list of integers and integer k
- Output: the kth smallest number in the set of integers.

```
function Selection(a[1…n],k)
    if n==1:
        return a[1]
    pick a random integer in the list v.
    Split the list into sets SL, Sv, SR.
    if $k \leq |SL|$:
        return Selection(SL,k)
    if $k \leq |SL|+|Sv|$:
        return v
    else:
        return Selection(SR, $k-|SL|-|Sv|$)
```

Selection (Runtime)

- Input: list of integers and integer k
- Output: the kth smallest number in the set of integers.

```
function Selection(a[1…n],k)
    if n==1:
        return a[1]
    pick a random integer in the list v.
    Split the list into sets SL, Sv, SR.
    if k≤|SL|:
        return Selection(SL,k)
    if k≤|SL|+|Sv|:
        return v
    else:
        return Selection(SR, k-|SL|-|Sv|)
```

Selection (Runtime)

- The runtime is dependent on how big are |SL| and |SR|.

- If we were so lucky as to choose v to be close to the median every time, then |SL|≈|SR|≈ $n/2$. And so, no matter which set we recurse on,
$$T(n) = T\left(\frac{n}{2}\right) + O(n)$$
- And by the Master Theorem:

Selection (Runtime)

- The runtime is dependent on how big are |SL| and |SR|.

- Conversely, if we were so unlucky as to choose v to be the maximum (resp. minimum) then |SL| (resp. |SR|) = n-1 and

$$T(n) = T(n-1) + O(n)$$

- Which is ………….?

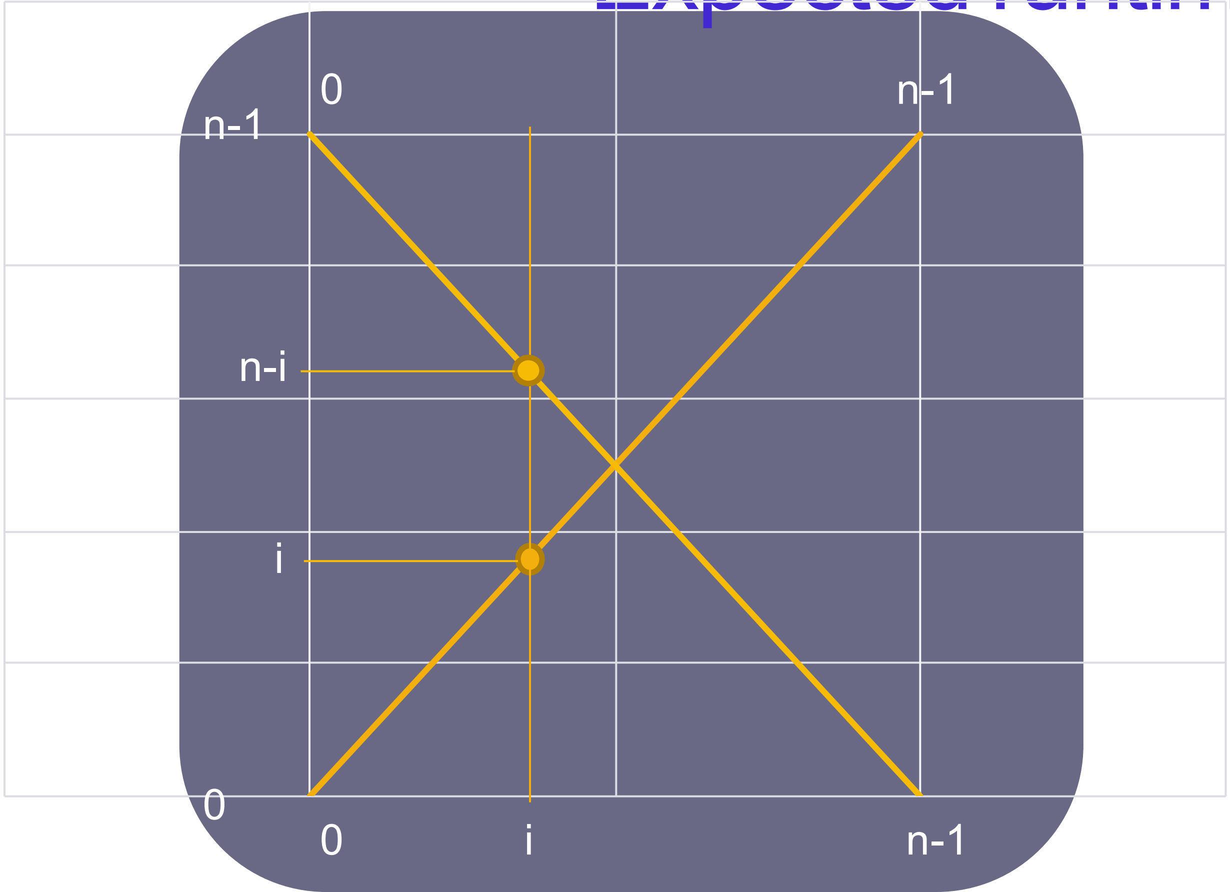
Selection (Runtime)

- The runtime is dependent on how big are |SL| and |SR|.

- Conversely, if we were so unlucky as to choose v to be the maximum (resp. minimum) then |SL| (resp. |SR|) = n-1 and
$$T(n) = T(n-1) + O(n)$$
- Which is $O(n^2)$, worse than sorting then finding.

- So is it worth it even though there is a chance of having a high runtime?

Expected runtime

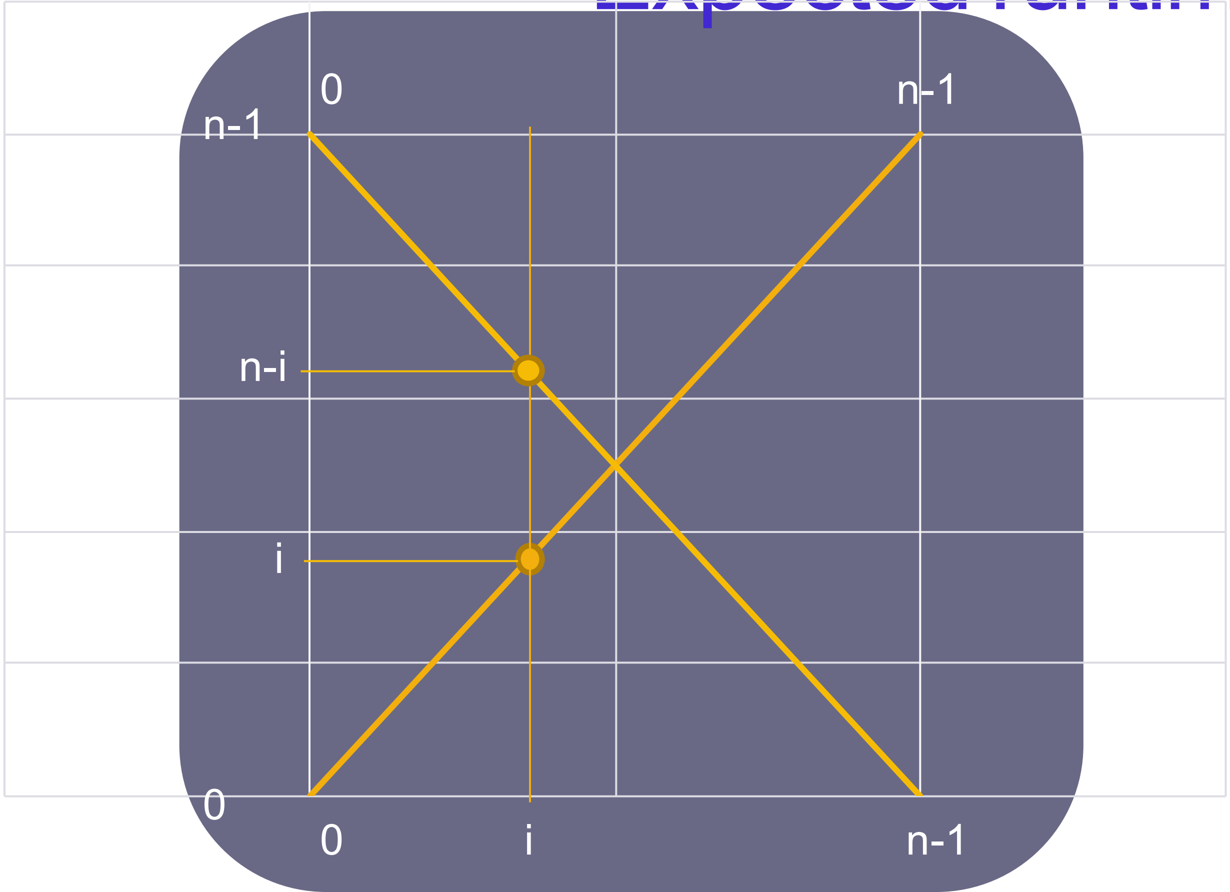


If you randomly select the ith element, then your list will be split into a list of length i and a list of length n-i.

So when we recurse on the smaller lists, it will take time proportional to

$$\max(i, n-i)$$

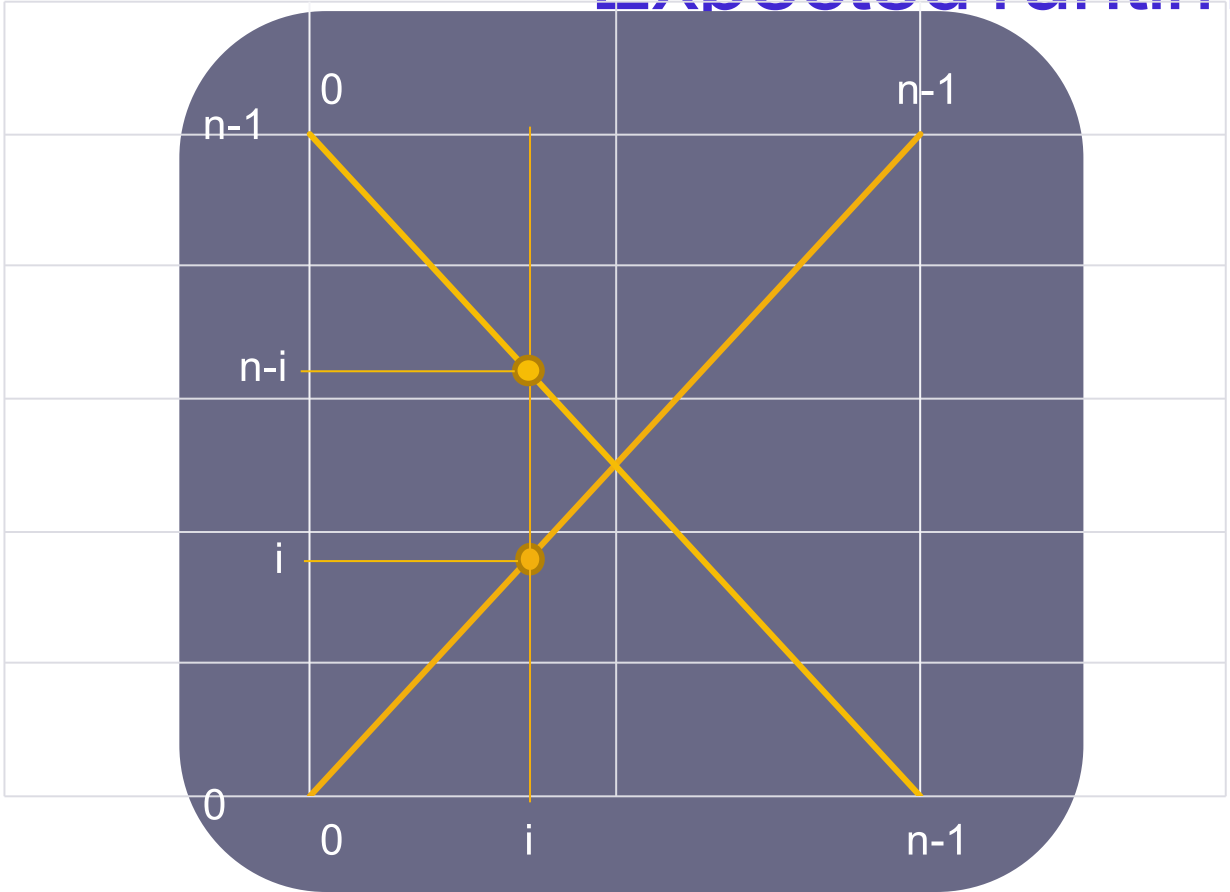
Expected runtime



Clearly, the split with the smallest maximum size is when i=n/2

and worst case is i=n or i=1.

Expected runtime

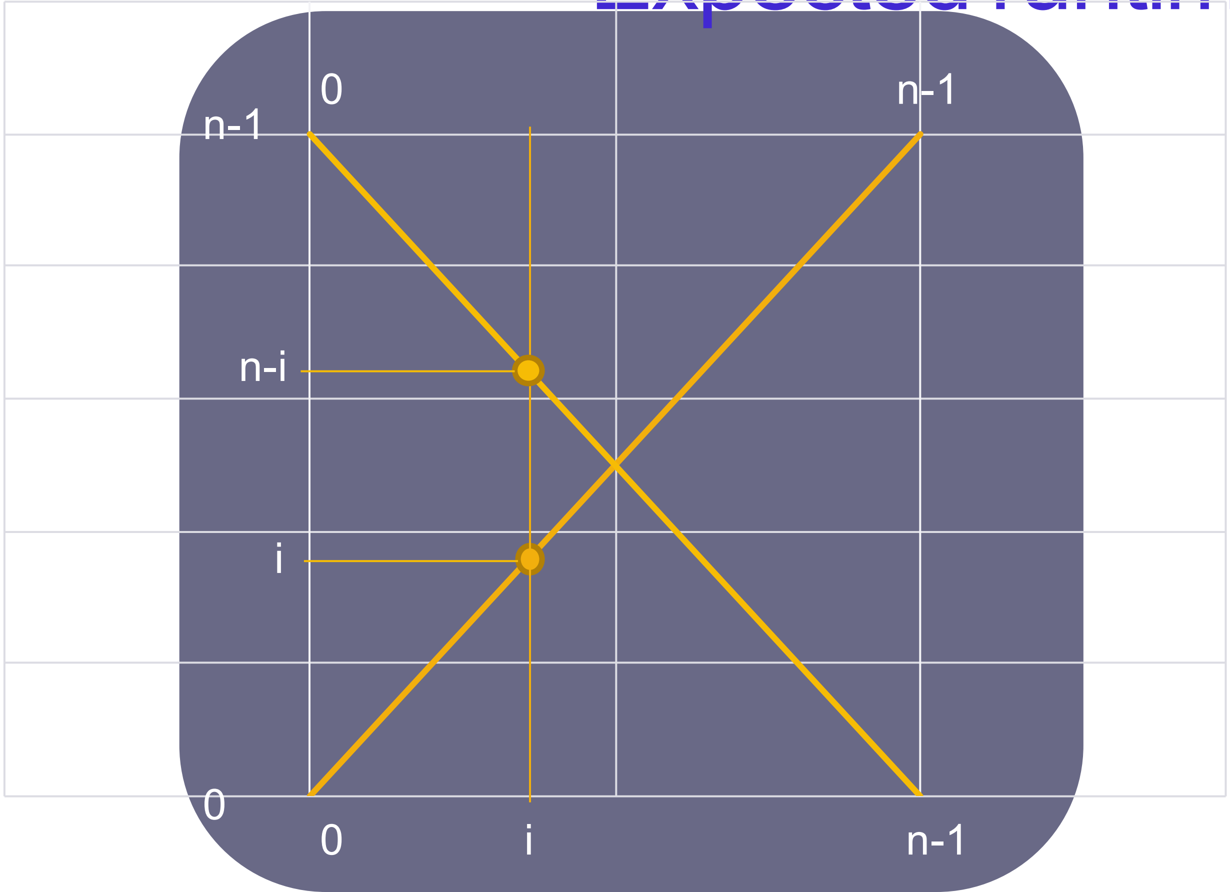


What is the expected runtime?

Well what is our random variable?

For each input and sequence of random choices of pivots, The random variable is the runtime of that particular outcome.

Expected runtime

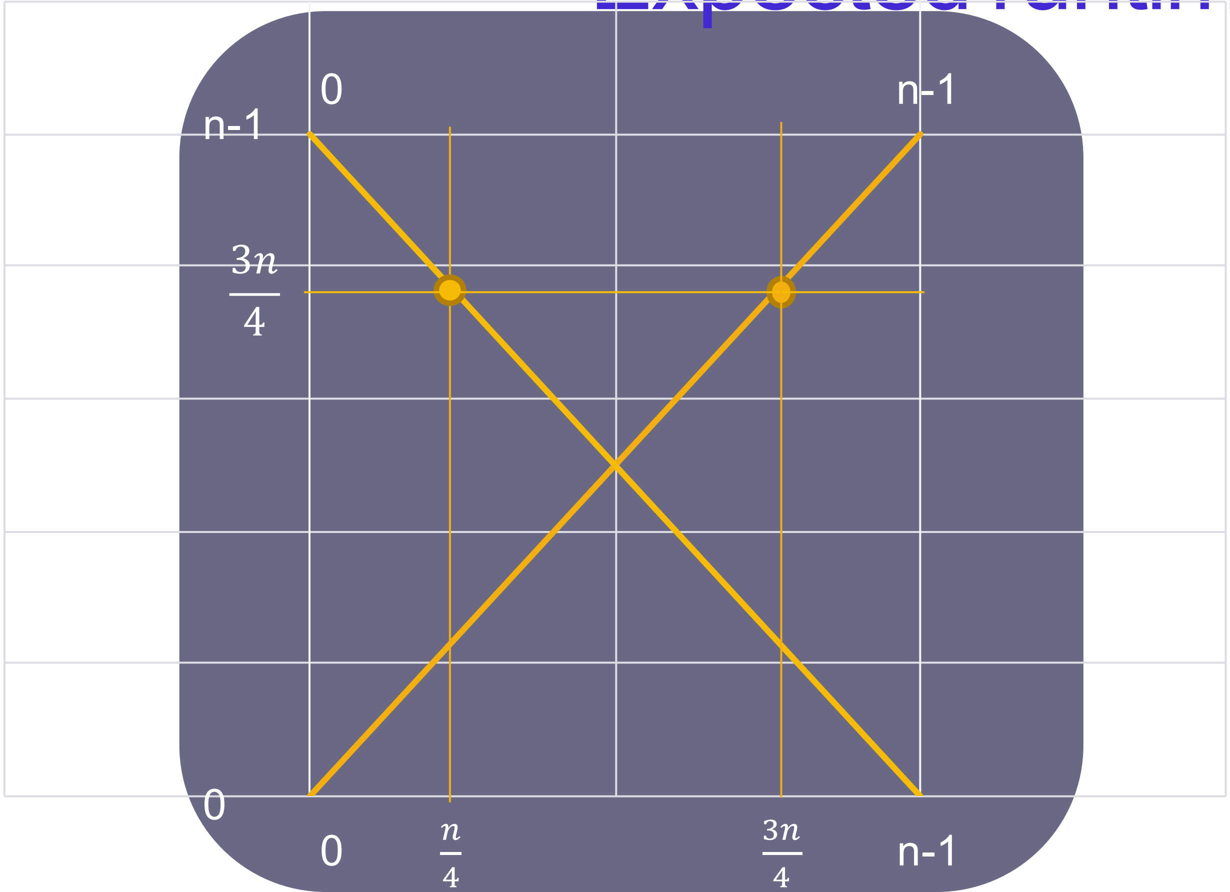


So if we want to find the expected runtime, we must sum over all possibilities of choices.

Let $ET(n)$ be the expected runtime. Then

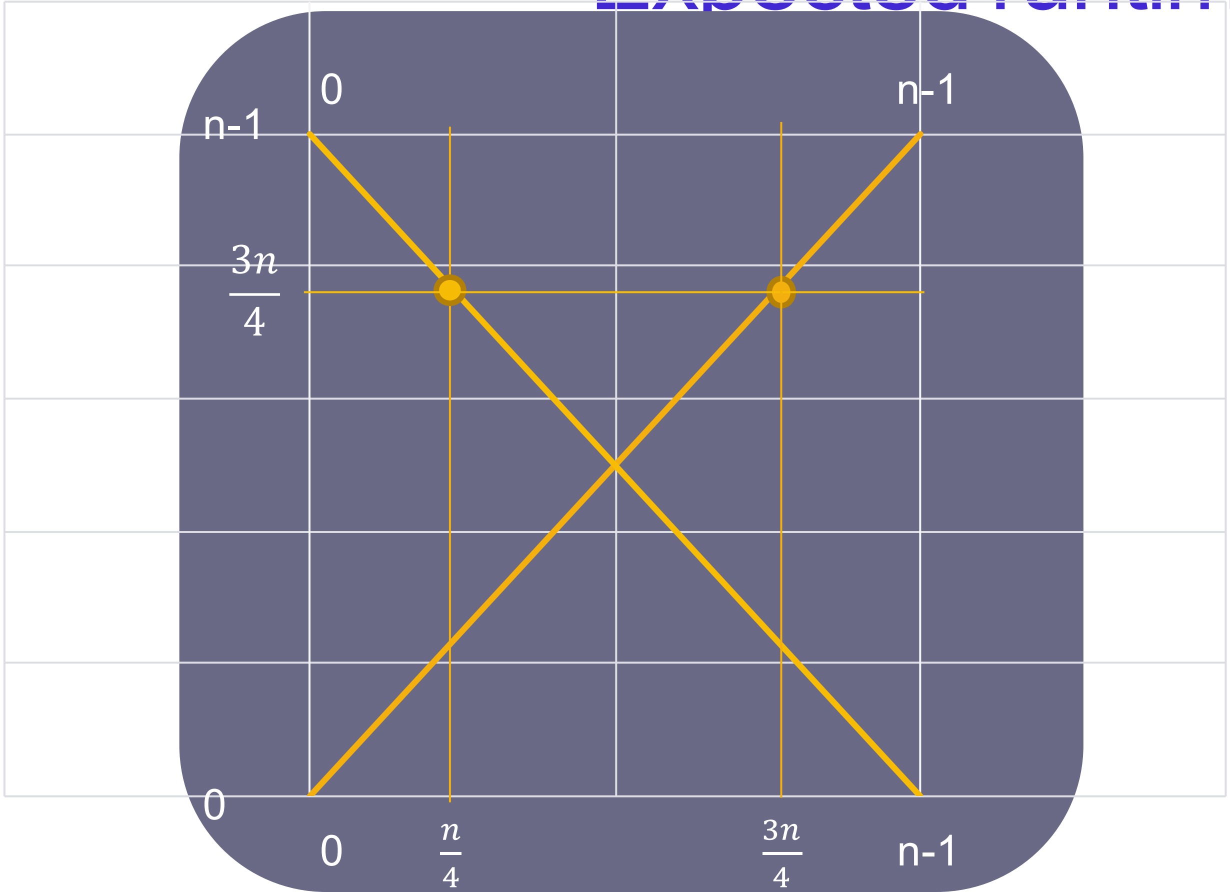
$$ET(n) = \frac{1}{n}\sum_{i=1}^{n} ET(\max(i, n-i)) + O(n)$$

Expected runtime



What is the probability of choosing a value from 1 to n in the interval $\left[\frac{n}{4}, \frac{3n}{4}\right]$ if all values are equally likely?

Expected runtime



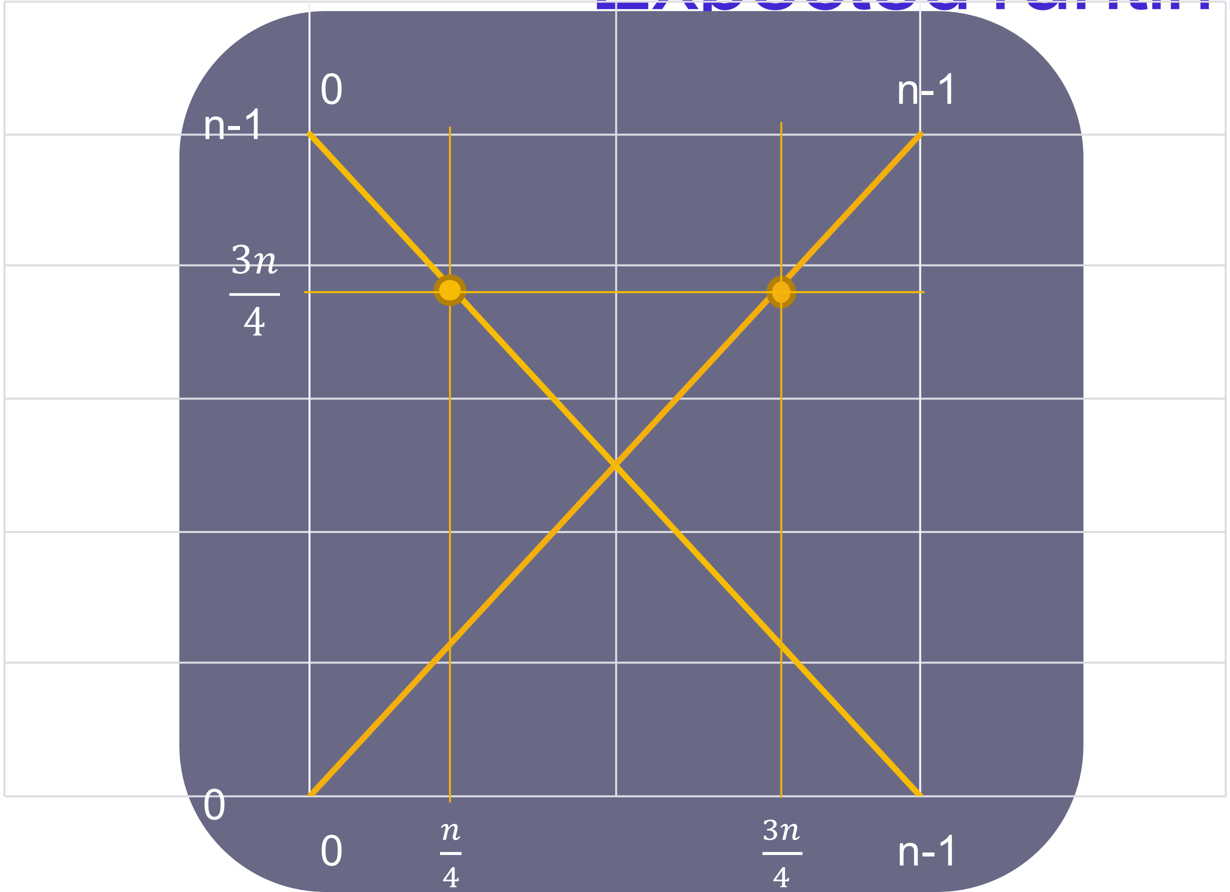
If you did choose a value between n/4 and 3n/4 then the sizes of the subproblems would both be $\leq \frac{3n}{4}$

Otherwise, the subproblems would be $\leq n$

So we can compute an upper bound on the expected runtime.

$$ET(n) \leq \frac{1}{2} ET\left(\frac{3n}{4}\right) + \frac{1}{2} ET(n) + O(n)$$

Expected runtime



$$ET(n) \leq \frac{1}{2}ET\left(\frac{3n}{4}\right) + \frac{1}{2}ET(n) + O(n)$$

$$ET(n) \leq ET\left(\frac{3n}{4}\right) + O(n)$$

Plug into the master theorem with a=1, b=4/3, d=1

$a<b^d$ so

$$ET(n) \leq O(n)$$

quicksort

- What have we noticed about the partitioning part of Selection?

- After partitioning, the “pivot” is in its correct position in sorted order.

- Quicksort takes advantage of that.

Quicksort divide and conquer

- Let’s think about selection in a divide and conquer type of way.

- Break a problem into similar subproblems
  - Split the list into two sublists by partitioning a pivot
- Solve each subproblem recursively
  - recursively sort each sublist
- Combine
  - concatenate the lists.

Quicksort divide and conquer

- procedure quicksort(a[1…n])
  - if n≤1:
    - return a
  - set v to be a random element in a.
  - partition a into SL,Sv,SR
  - return quicksort(SL)∘Sv∘ quicksort(SR)

Quicksort (runtime)

```
procedure quicksort(a[1…n])
    if $n \le 1$:
        return a
    set v to be a random element in a.
    partition a into SL,Sv,SR
    return quicksort(SL)∘Sv∘ quicksort(SR)
```

Quicksort (runtime)

$$ET(n) = \frac{1}{n}\left(\sum_{i=1}^{n} ET(n-i) + ET(i) + O(n)\right)$$

Bounding quicksort time

- However we break up inputs into subsets, at most cn total time per recursive levels.
- So need to bound depth of recursion.
- Claim: With high probability the depth of recursion is O(log n).

Why is quicksort quick?

- Good PR.

- But MergeSort also O(n log n) comparisons.

Factors outside number of steps

- What other factors contribute to how long algorithms take on actual machines?   (Architecture, OS)

Factors outside number of steps

- What other factors contribute to how long algorithms take on actual machines? (Architecture, OS)

- ``Locality of reference'': when data accessed, moved to cache. Often moved in consecutive blocks. When it is accessed frequently, doesn't get evicted from cache.
- Quicksort: data in common subproblems moved to be close together. Can sort in place, rather than repeatedly merging.

Selection (Deterministic)

- Sometimes this algorithm we have described is called quick select because generally it is a very practical linear expected time algorithm. This algorithm is used in practice.
- For theoretic computer scientists, it is unsatisfactory to only have a randomized algorithm that could run in quadratic time.
- Blum, Floyd, Pratt, Rivest, and Tarjan have developed a deterministic approach to finding the median (or any k^{th} biggest element.)

- They use a divide and conquer strategy to find a number close to the median and then use that to pivot the values.

Selection (Deterministic)

- The strategy is to split the list into sets of 5 and find the medians of all those sets. then find the median of the medians using a recursive call T(n/5).
- Then partition the set just like in quickselect and recurse on SR or SL just like in quickselect.

Median of medians

- MofM(L,k)
  - If L has 10 or fewer elements:
    - Sort(L) and return the kth element
  - Partition L into sublists S[i] of five elements each
  - For $i = 1, \ldots n/5$
    - $m[i]$ =MofM(S[i],3)
  - M = MofM([$m[1], \ldots, m[n/5]$],$n/10$)
  - ???

Median of medians

- MofM(L,k)
  - If L has 10 or fewer elements:
    - Sort(L) and return the kth element
  - Partition L into sublists S[i] of five elements each
  - For $i = 1, \ldots n/5$
    - $m[i]$ =MofM(S[i],3)
  - M = MofM($[m[1], \ldots, m[n/5]]$,$n/10$)
  - Split the list into sets SL, SM, SR.
  - if k≤|SL|:
    - return ~~Selection~~ MofM(SL,k)
  - if k≤|SL|+|Sv|:
    - return v
  - else:
    - return ~~Selection~~ MofM(SR, k-|SL|-|Sv|)

Selection (Deterministic)

- By construction, it can be shown that |SR|<7n/10 and |SL|<7n/10 and so no matter which set we recurse on, we have

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$

- You cannot use the master theorem to solve this, but you can use induction to show that if $T(n) \leq cn$ for some c, then $T(n) \leq cn$.
- And so we have a linear time selection algorithm!!!!!

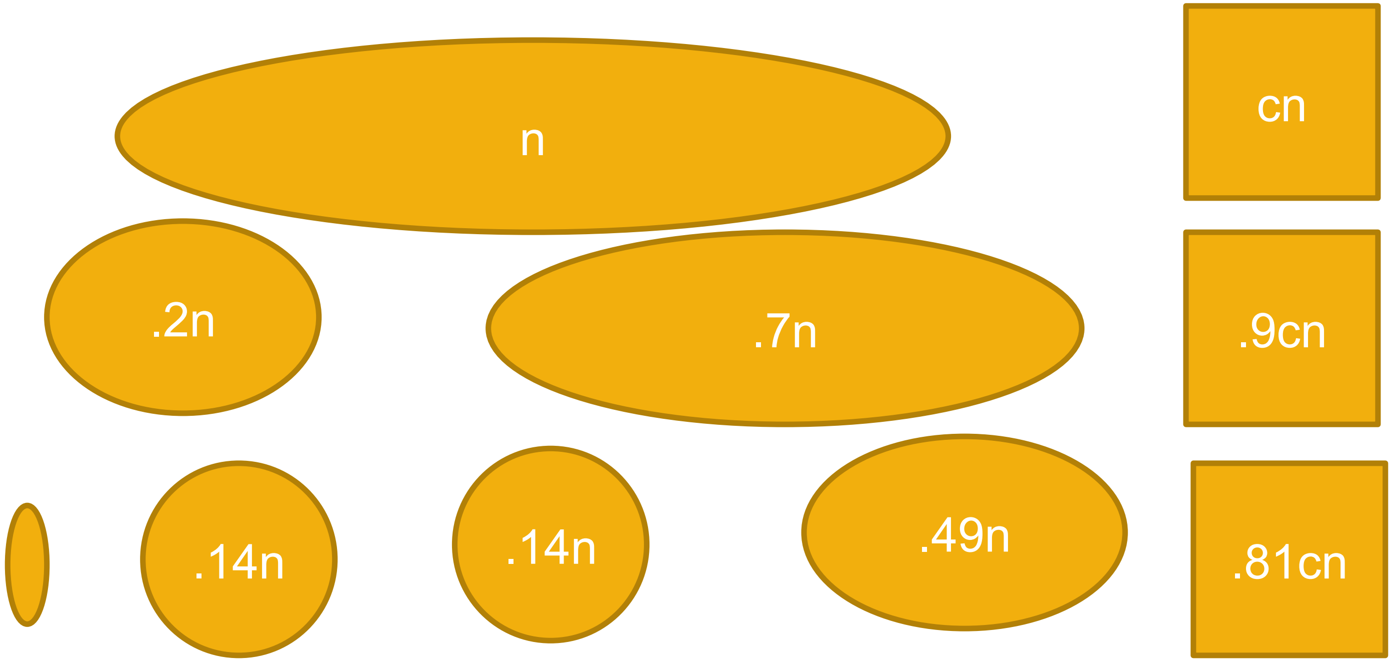
Selection (Deterministic)

- We showed that M is between 3n/10 and 7n/10 in sorted order. So both |SR|<7n/10 and |SL|<7n/10 and so no matter which set we recurse on, we have

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$

- You cannot use the master theorem to solve this, but you can use induction to show that if $T(n) \leq cn$ for some c, then $T(n) \leq cn$.

Time analysis



Total time

- Top heavy: Work decreasing geometically as we go down, total cn (1 +.9 + (.9)^2+ (.9)^3….) =10cn