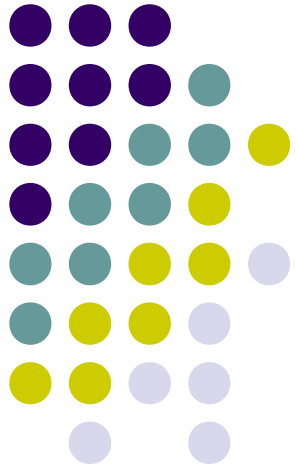
COMP20003
Algorithms and Data Structures
Mergesort

Nir Lipovetzky

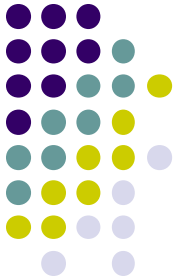
Department of Computing and Information Systems

University of Melbourne

Semester 2

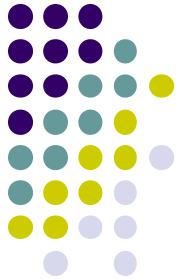


Quicksort: Summary



- http://www.youtube.com/watch?v=ywWBy6J5gz8

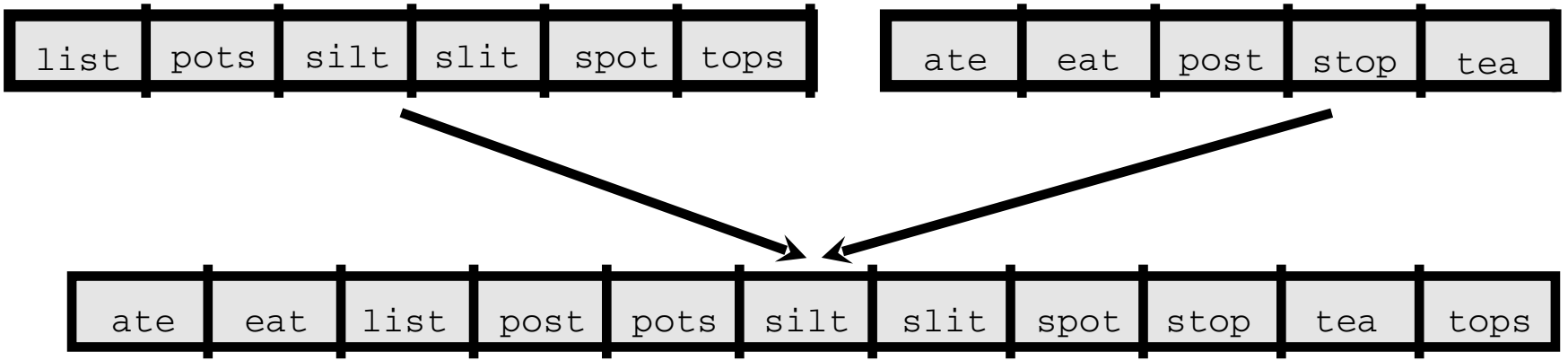
Mergesort



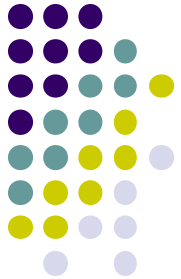
- Skiena Chapter 4.5

Merging

- We have two lists (stored as linked lists or arrays), each already in sorted order.
- We would like to merge them into one sorted list that includes every element.



How do you merge?

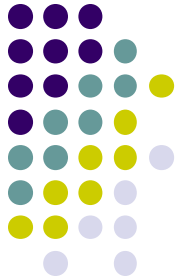


- 2 linked lists or arrays
  - Two pointers (or indices): to smallest element.
  - Compare elements pointed to.
  - Output smallest and move pointer.
- How many comparisons?
- Code…

Merge Code

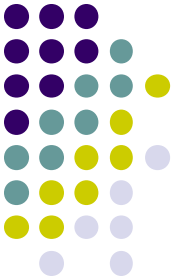
```
merge(item C[], item A[], item B[],
      int n, int m) /* n is size of A, m size of B */
{









}
```

Merge Code

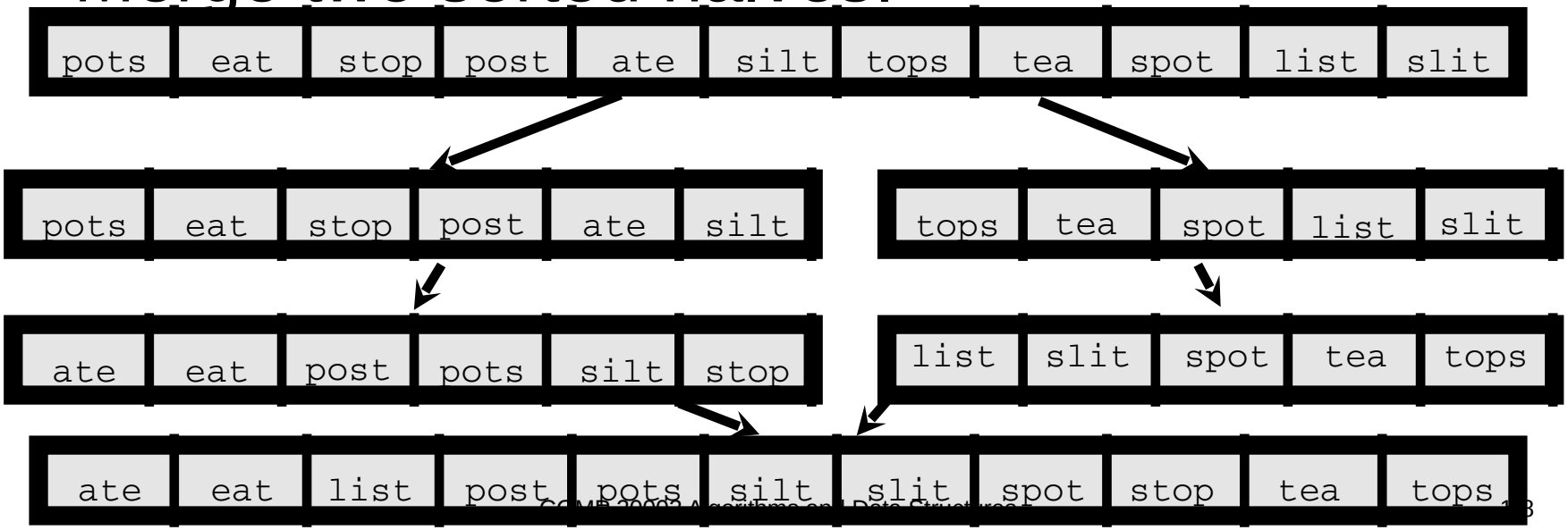


```
merge(item C[], item A[], item B[],
      int n, int m) /* n is size of A, m size of B */
{
   int i,j,k; for(i=0,j=0,k=0;k<n+m;k++)
   {
     /* shortcut at the end of A or B*/
     if(i==n) {C[k]=B[j++];continue;}
     if(j==m) {C[k]=A[i++];continue;}
     if(A[i}<=B[j]) C[k] = A[i++];
     else C[k] = B[j++];
    }
}




}
```

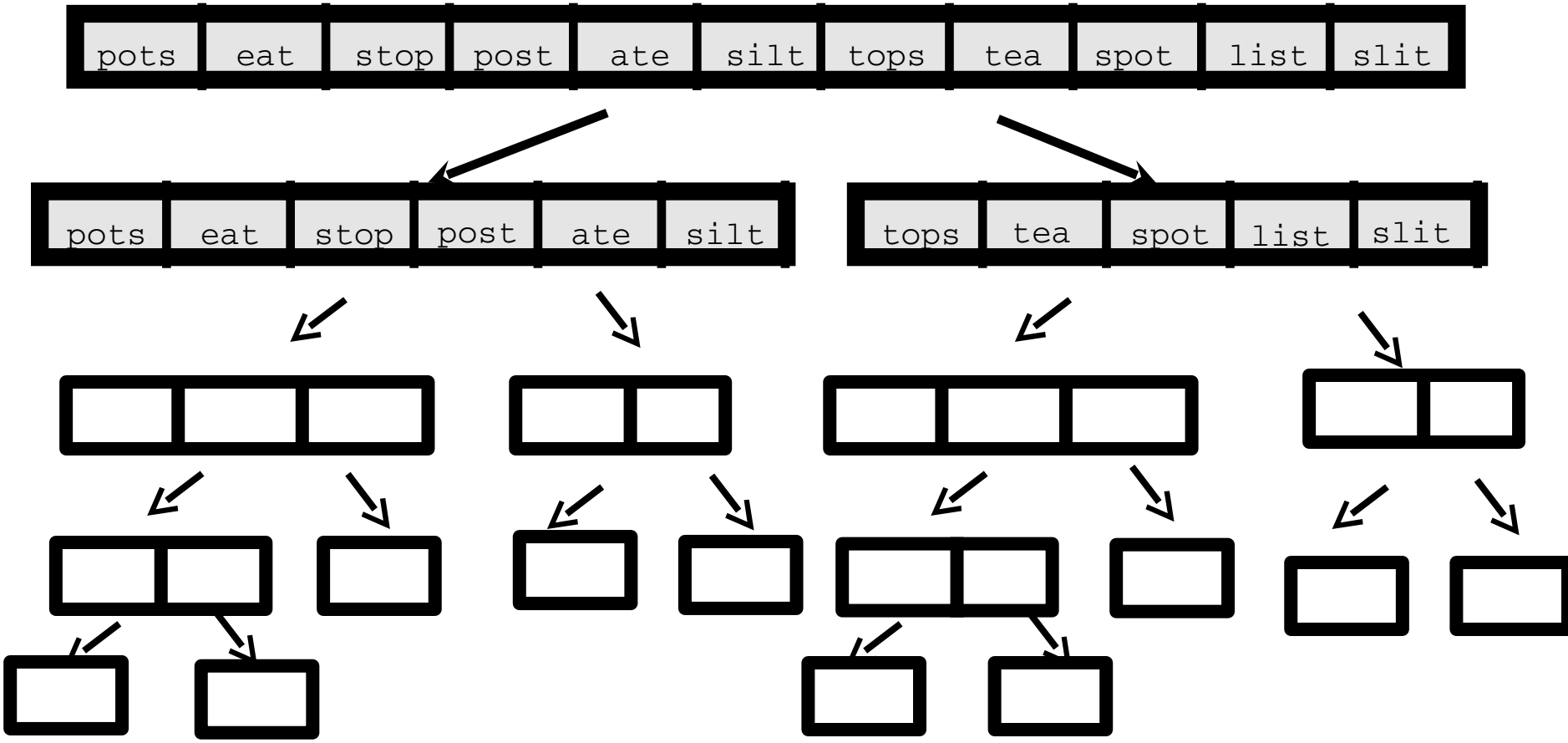
Sorting using the merge operation



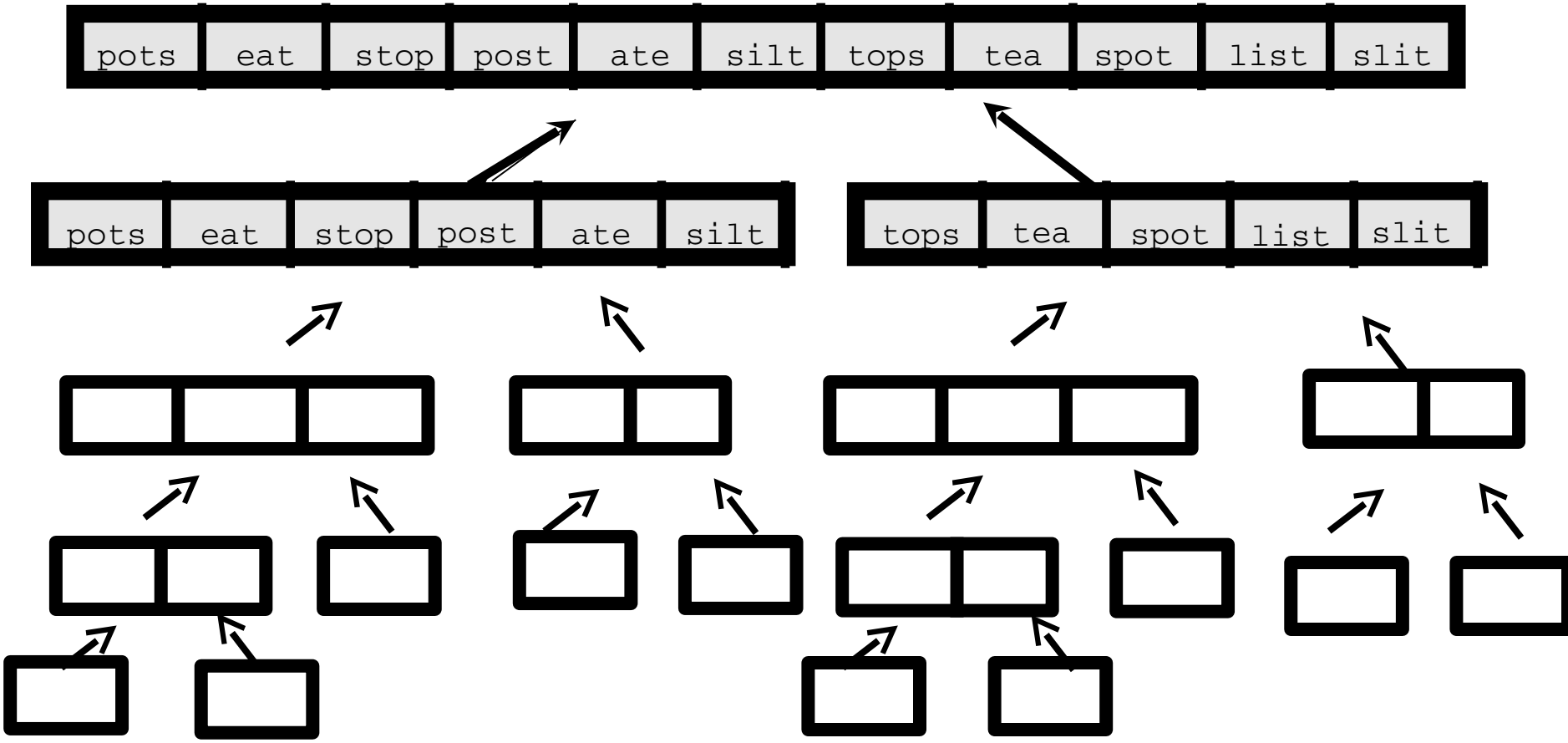
- If list has one element, return.
- Split list into two equal-sized pieces (recursively, until singleton)
- Sort each half.
- Merge two sorted halves.



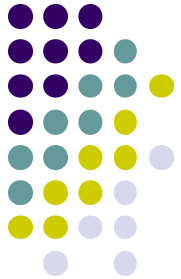
Mergesort: splitting



Mergesort: merging

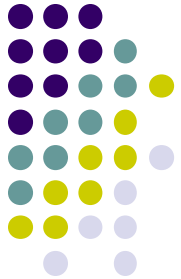


Mergesort: topdown (recursive)



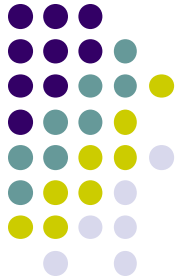
```
main() {/* code */ mergesort(A,0,n-1); /* more code */

mergesort(A,first,last)
{   int i;
    item B[], item C[];
    mid = (int)(last-first+1)/2;
    for(i=0;i<mid;i++) B[i] = A[i];
    for(i=mid;i<=last;i++) C[i-mid] = A[i];
    B = mergesort(B,0,mid-1);
    C = mergesort(C,0,mid-1);
    A = merge(B,C);
}
```

Analyzing mergesort



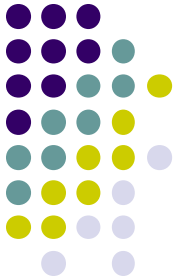
- We are concerned with:
  - Accuracy
    - Does mergesort work?
    - Is it stable?
  - Efficiency
    - Does it take extra space? How much?
    - Analyze time efficiency using recurrences.

Recurrences



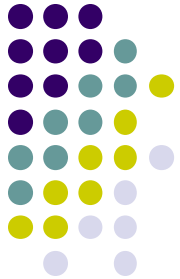
- Recurrence relation (mathematical defn):
  - an equation that recursively defines a sequence.
  - once one or more initial terms are given.
  - each further term of the sequence is defined as a function of the preceding terms.
- As we did for the Fibonacci numbers.

Mergesort recurrences



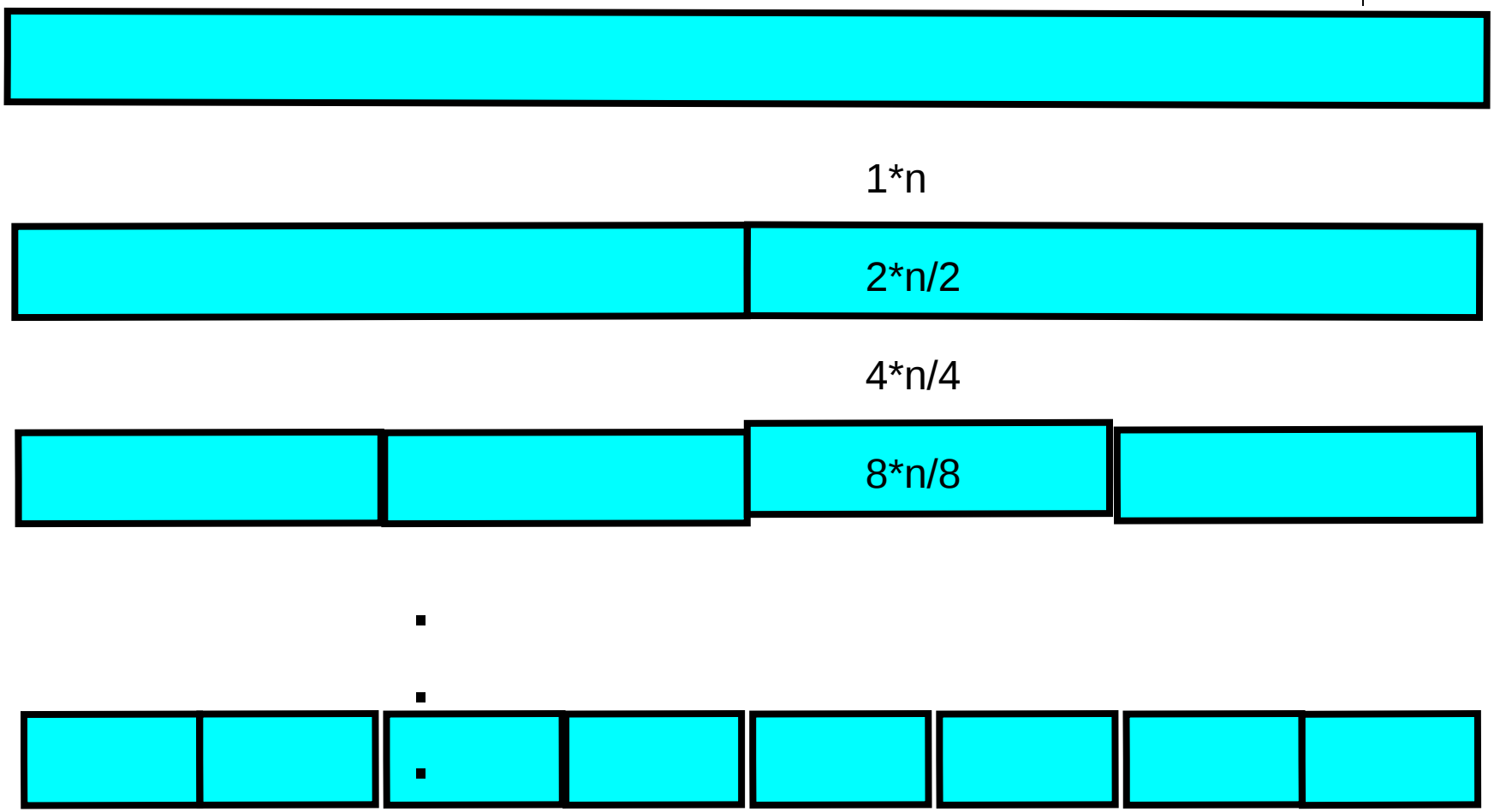
- Recurrence for number of comparisons:
- Cost of sorting n items =
  - 2*Cost of sorting n/2 items + merge n items
  - $C(n) = 2C(n/2) + n-1$ (worst case)
  - $C(1) = 0$

Solving recurrence

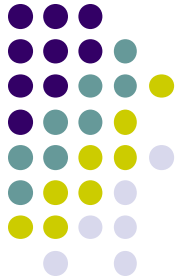


- Approximate n as power of 2:
  - $C(n) = 2C(n/2) + n-1$
  - $= 2[2C(n/4) + (n/2-1)] + (n-1)$
  - $= 4C(n/4) + (n-2) + (n-1)$
  - $= 8C(n/8) + (n-4) + (n-2) + (n-1)$
  - … $\log_2 n$ splits
  - $2^{\log n} + n + n + \ldots$<log n times> n
  - $\approx n \log n$

Intuition



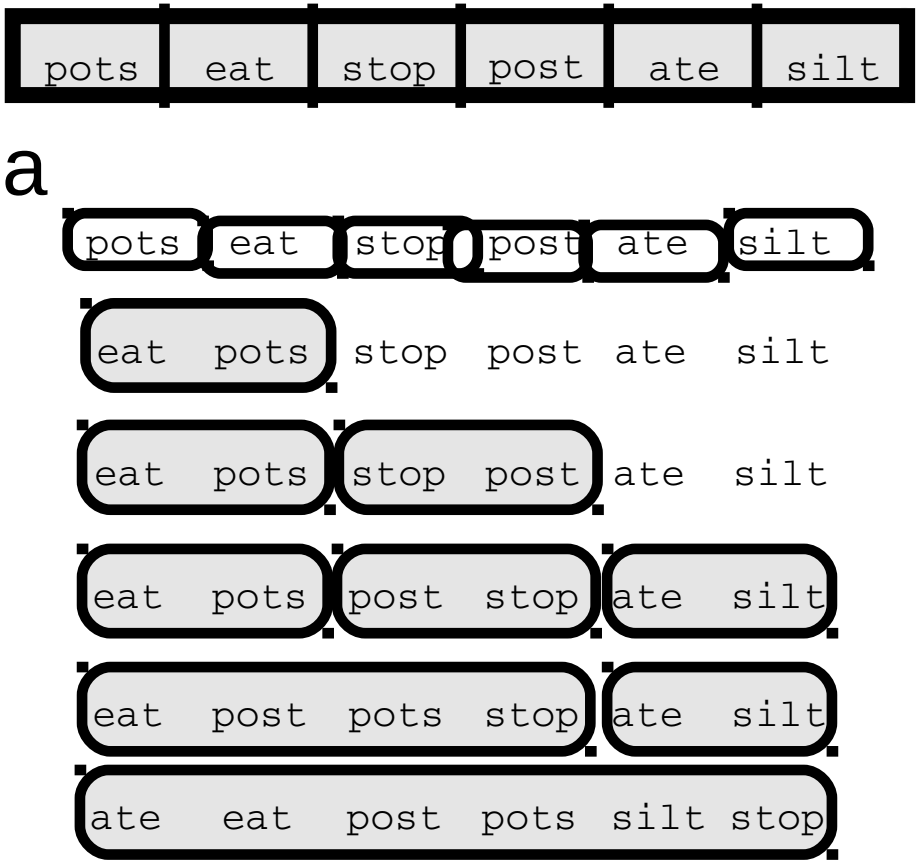
Mergesort: Top-down (recursive)



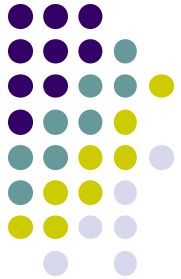
- Top-down mergesort works well with arrays.
- Also with linked lists (with a pointer to find midpoint of list).
- Worst case O(n log n).
- Average case O(n log n)
- Stable?
- Requires O(n) extra space.

Bottom-up mergesort

- Break list into *n* singleton lists.
- Insert single lists into a queue.
- deQueue the first two items, merge them, and enQueue them.
- Merged Items go at the end

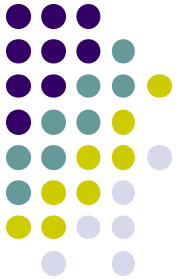


Mergesort: Implementation



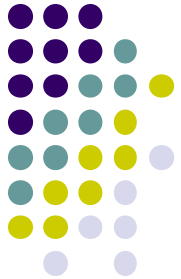
- Top-down mergesort (recursive).
- Bottom-up mergesort (iterative).
- https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Mergesort: Analysis



- Analysis similar for recursive and non.
  - $\Theta(n \log n)$.
  - Stable.
  - Reliable, and work with both arrays and lists.
  - Can sort huge files on disk. ☺
    - Use disk fetching just the portions of data you need
- Would be the perfect sort, except that;
  - Arrays require O(n) extra space.
  - Slower than quicksort ☹

Mergesort: Summary



- http://www.youtube.com/watch?v=XaqR3G_NVoo