Randomized Algorithms

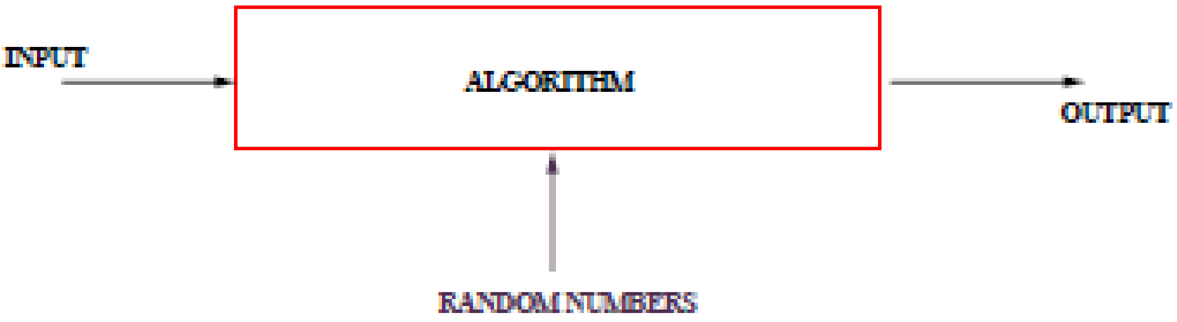
Introduction

Types

- Las Vegas (Ex. Randomized Quick Sort)
- Monte Carlo (Ex. Karger's Min Cut)

Introduction

- A randomized algorithm is one that receives, in addition to its input data, a stream of random bits that it can use for the purpose of making random choices.



- Even for a fixed input, different runs of a randomized algorithm may give different results.

Example

- Search for an element '*x*' in an array *A*.

1. search (int *A*[], int *x*)
2. for i = 0 to *A*.*length*
3. check if element exists.
4. end for

Simple Algorithm

search (int *A*[], int *x*)

while(1)

Randomly select one element out of *A*.*length* elements.

until '*x*' is found

end

Randomized Algorithm

Contd…

- An extremely important tool for the construction of algorithms in a wide range of applications.
- Principal advantages:
  - Relatively smaller execution time or space requirement than that of the best known deterministic algorithm for the same problem.
  - Extremely simple to understand and to implement.
- Running time of randomized algorithms is given as an expected values.

Types

- Las-Vegas
  - Always gives correct results or it informs about the failure.
  - Running time is a random variable.
  - Example: Randomized quicksort, where the pivot is chosen randomly, but the result is always sorted.
- Monte Carlo
  - May produce an incorrect result with some probability.
  - Running time is fixed.
  - Example: Karger's Min Cut, to get a min-cut for a given graph G with some probability.

Example – Search (int *A*[], int *x*)

```
searchLasVegas(int A[], int x)
    while(true)
        Randomly select one
          element out of A.length
          elements.
        if (found)
            return true
    end
```

```
searchMonteCarlo(int A[], int x)
    i = 0; flag = false
    while(i <= <some number>)
        Randomly select one
          element out of A.length
          elements.
        i++;
        if (found)
            flag = true
    end
    return flag
```

QUICKSORT(A, p, r)

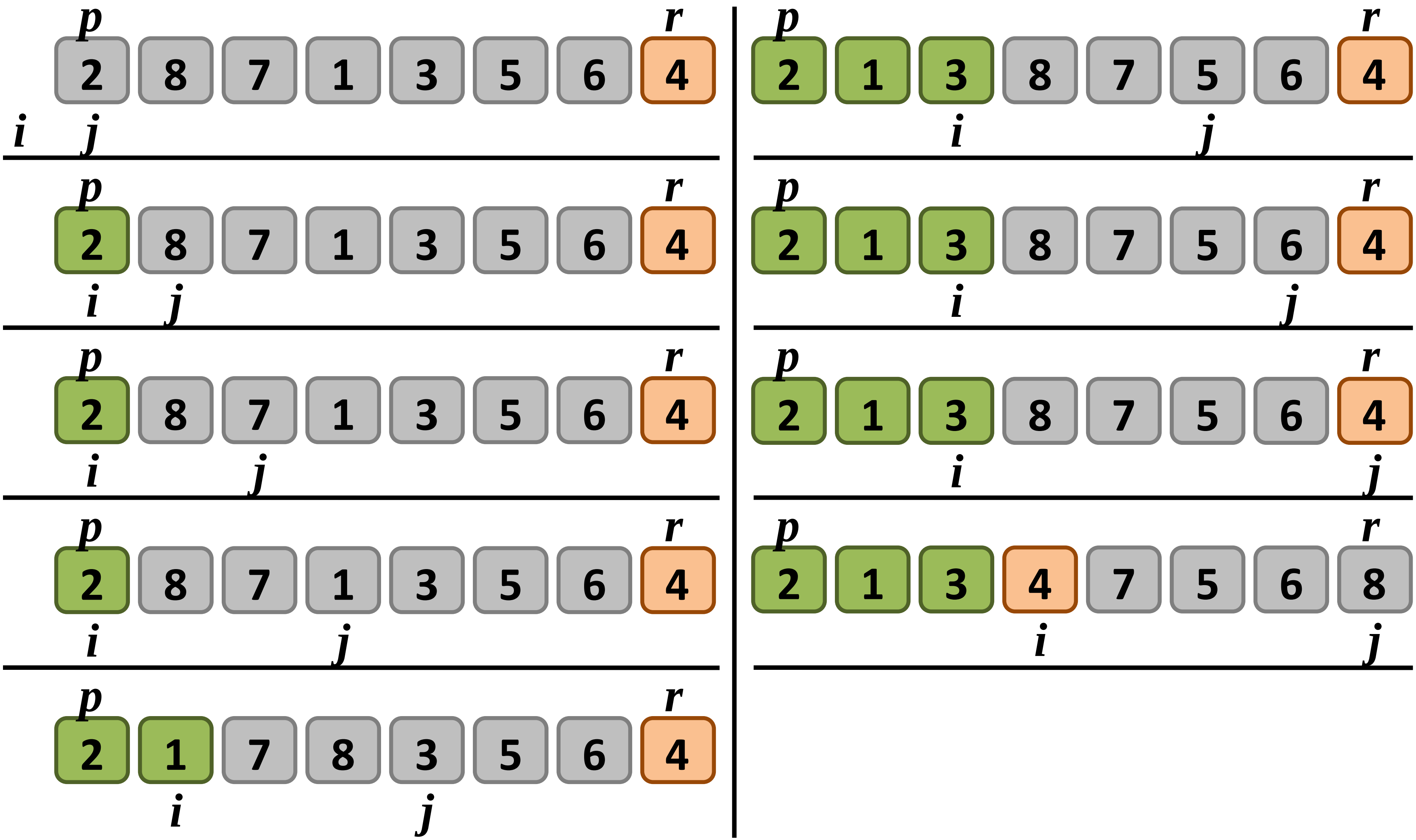
- QUICKSORT(A, p, r)

```
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q – 1)
    QUICKSORT(A, q + 1, r)
```

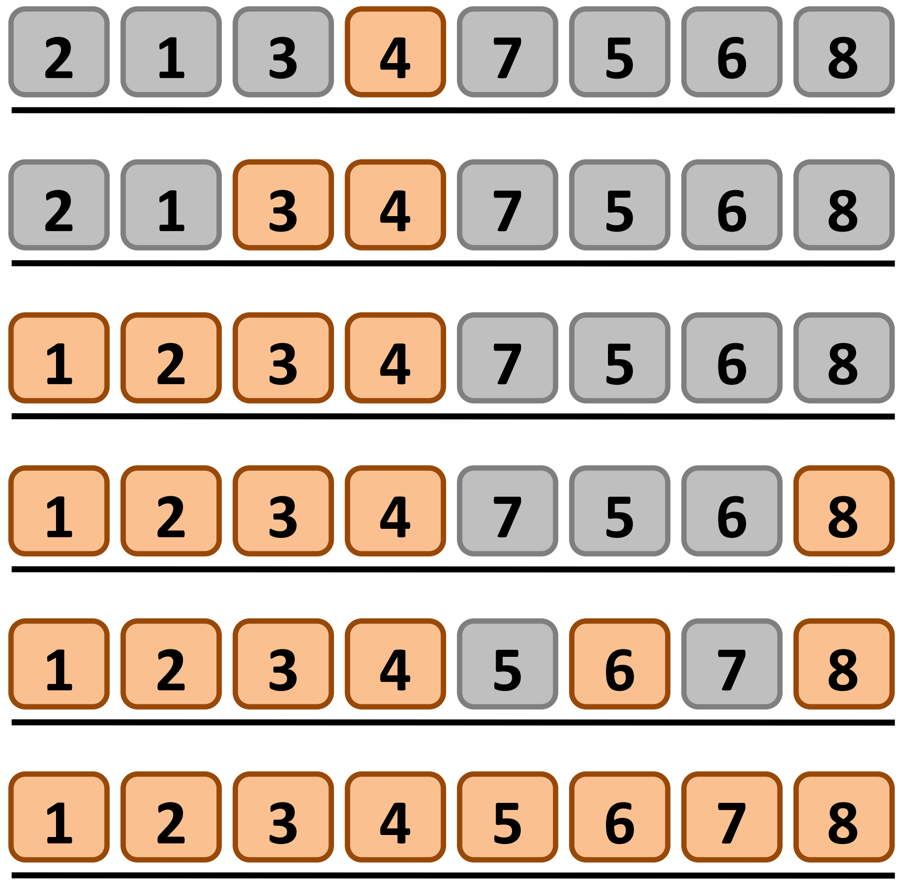
- To sort an array A with n elements, the first call to QUICKSORT is made with $p = 0$ and $r = n - 1$.

```
PARTITION(A, p, r)
  x = A[r]
  i = p – 1
  for j = p to r – 1
      if A[j] ≤ x
          i = i + 1
          Exchange A[i] with A[j]
  Exchange A[i + 1] with A[r]
  return i + 1
```

Example: 2, 8, 7, 1, 3, 5, 6, 4



Contd...



Randomized Quick Sort (Las Vegas)

- RANDOMIZED-QUICKSORT(A, p, r)

1. if $p < r$
2. $\quad q$ = RANDOMIZED-PARTITION(A, p, r)
3. $\quad$ RANDOMIZED-QUICKSORT(A, p, $q - 1$)
4. $\quad$ RANDOMIZED-QUICKSORT(A, $q + 1$, r)

- RANDOMIZED-PARTITION(A, p, r)

1. i = RANDOM(p, r)
2. Exchange $A[r]$ with $A[i]$
3. return PARTITION(A, p, r)

Example

0	1	2	3	4	5	6	7
5	3	8	9	4	7	6	1

- Generate a random number in between 0 and 7.
  - Let the number be 5.
  - Exchange $A[5]$ with $A[7]$

0	1	2	3	4	5	6	7
5	3	8	9	4	1	6	7

Contd...

	5	3	8	9	4	1	6	7
i	*pj*							*r*

	5	3	8	9	4	1	6	7
	pi	*j*						*r*

	5	3	8	9	4	1	6	7
	p	*i*	*j*					*r*

	5	3	8	9	4	1	6	7
	p	*i*		*j*				*r*

	5	3	8	9	4	1	6	7
	p	*i*			*j*			*r*

5	3	4	9	8	1	6	7
p		*i*			*j*		*r*

5	3	4	1	8	9	6	7
p			*i*			*j*	*r*

5	3	4	1	6	9	8	7
p				*i*			*jr*

5	3	4	1	6	7	8	9
p					*i*		*jr*

Contd…

5	3	4	1	6	7	8	9
5	6	4	1	3	7	8	9
1	3	4	5	6	7	8	9
1	3	4	5	6	7	8	9
1	3	6	5	4	7	8	9
1	3	4	5	6	7	8	9
1	3	4	5	6	7	8	9
1	3	4	5	6	7	8	9
1	3	4	5	6	7	8	9
1	3	4	5	6	7	9	8
1	3	4	5	6	7	8	9
1	3	4	5	6	7	8	9

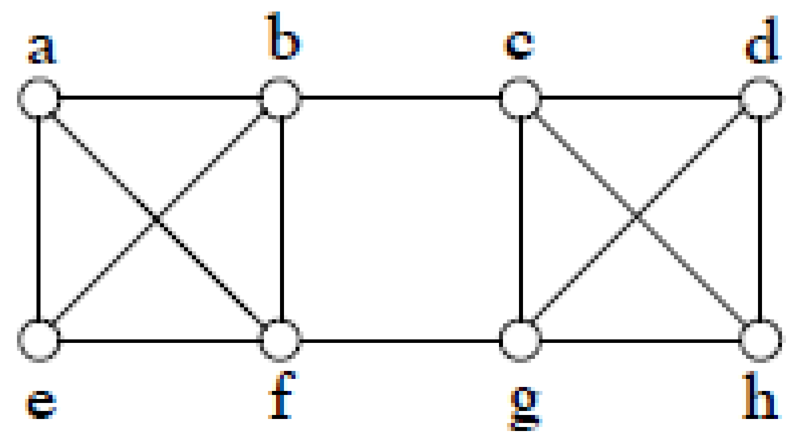
Exchanged

Final

Karger's Min-cut (Monte Carlo)

- Minimum cut of an undirected graph $G = (V,E)$ is a partition of the nodes into two groups V_1 and V_2, so that the number of edges between V_1 and V_2 is minimized.
  - $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$

Example:
Size of minimum cut is two with node partitions as $V_1 = \{a, b, e, f\}$ and $V_2 = \{c, d, g, h\}$.



Contd…

1. Repeat until just two nodes remain:
2. Pick an edge of G at random and collapse its two endpoints into a single node.

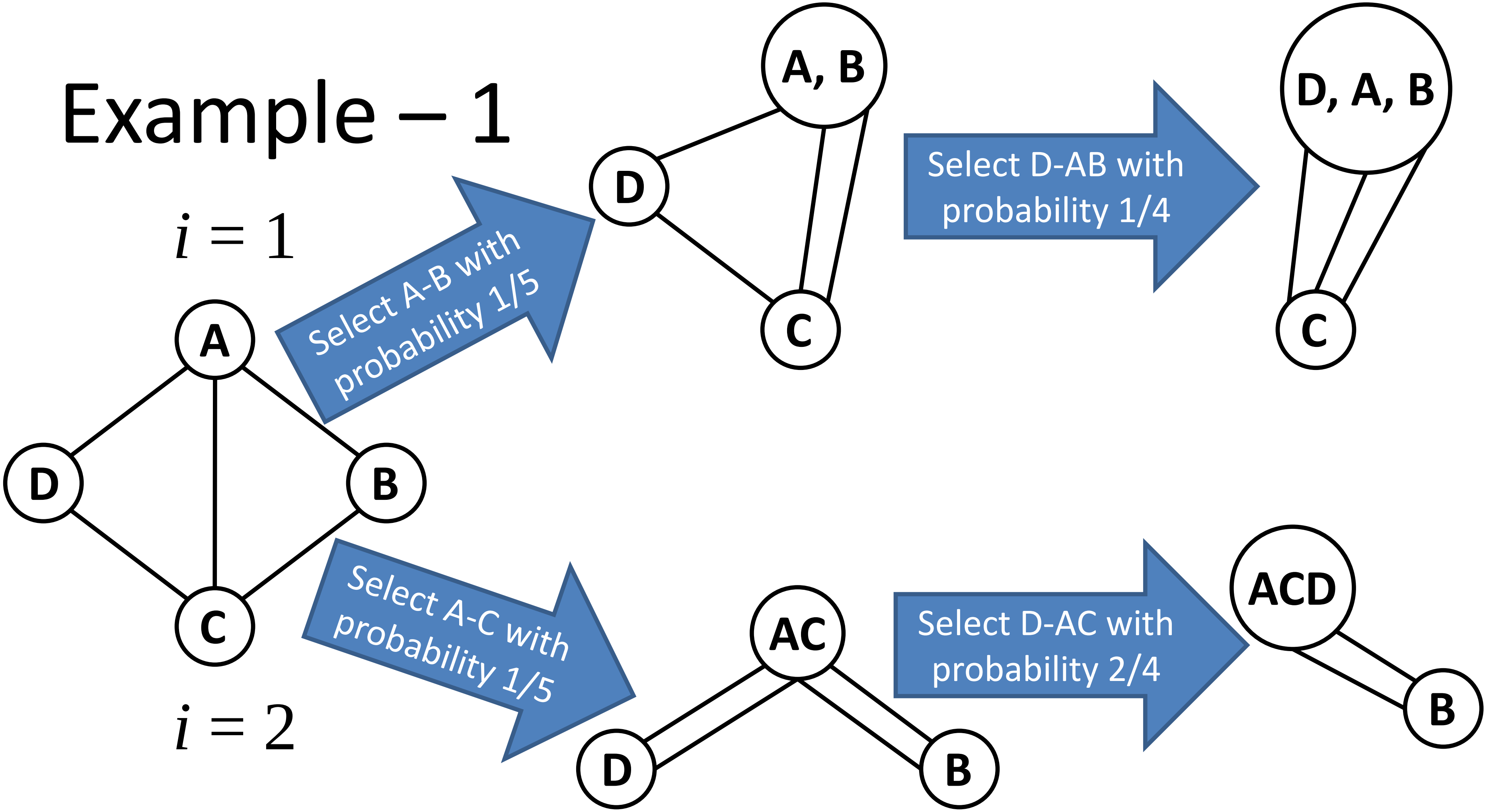
- For the two remaining nodes u_1 and u_2, set V_1 = {nodes in u_1} and V_2 = {nodes in u_2}

Karger's (basic algorithm)

- begin
- $i = 1$
- repeat
- repeat
- Take a random edge $(u,v) \in E$ in G
- replace u and v with the contraction u'
- until only 2 nodes remain
- obtain the corresponding cut result C_i
- $i = i + 1$
- until $i = m$
- output the minimum cut among $C_1, C_2, \ldots, C_m$.
- end

Example – 1

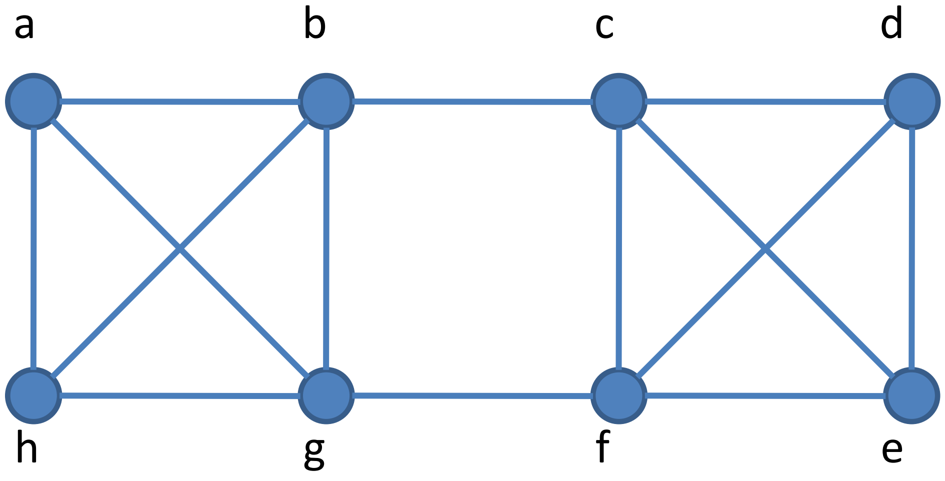
$i = 1$



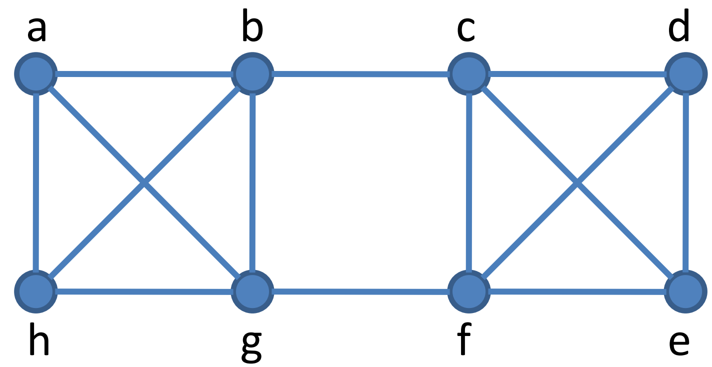
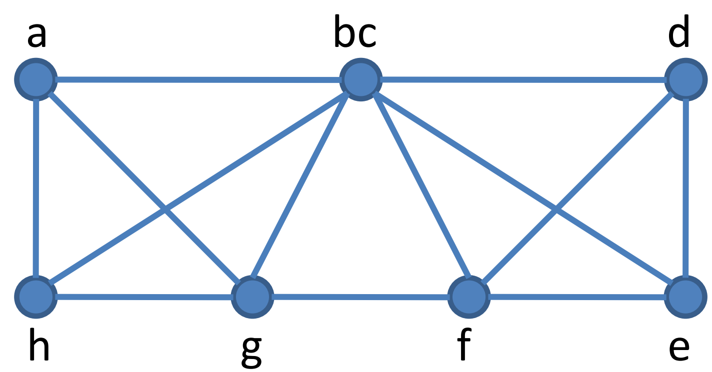
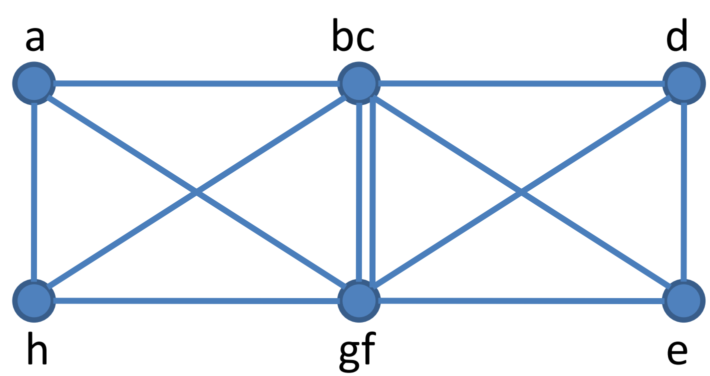
$i = 2$

i	C_i (# of edges)	Probability	Result is minimum C_i value (or C_i with higher probability). Min cut is 2 with vertex sets $\{A, C, D\}$ and $\{B\}$.
1	3	1/4 = 0.25	
2	2	2/4 = 0.5	

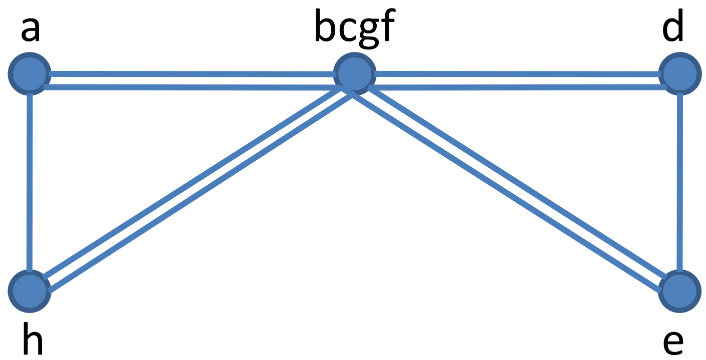
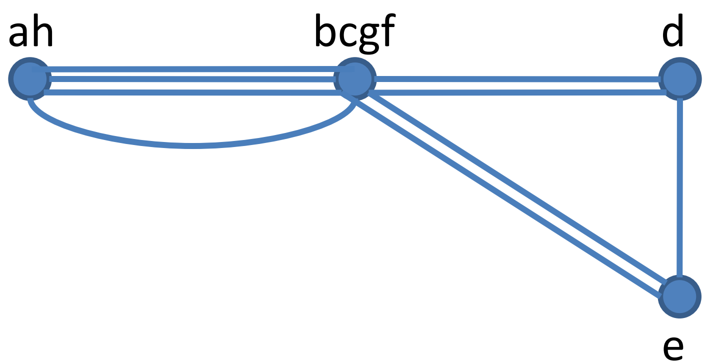
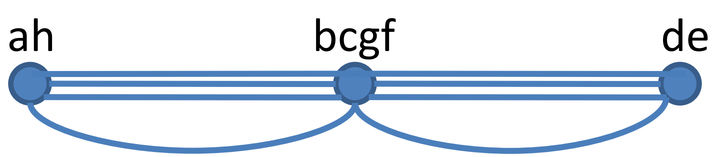
Example – 2



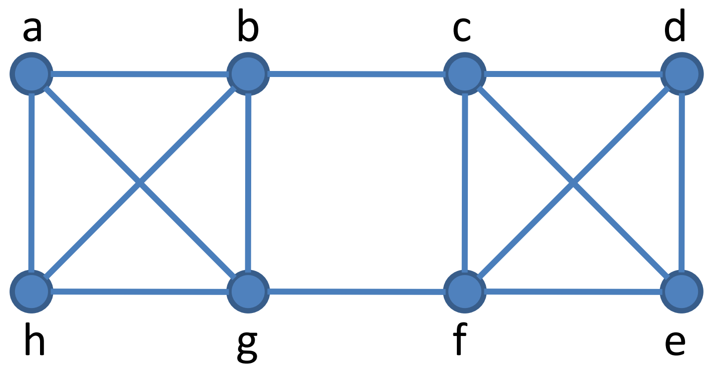
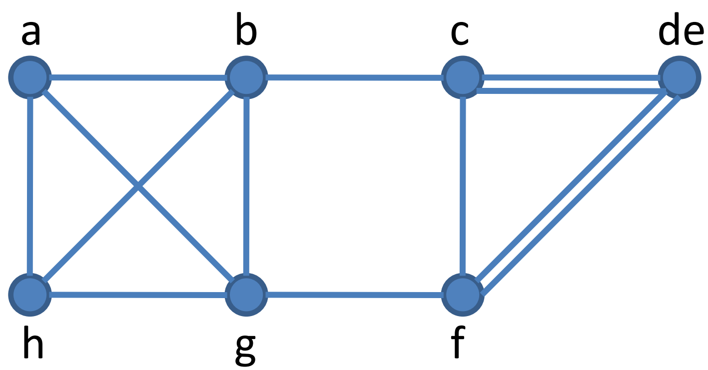
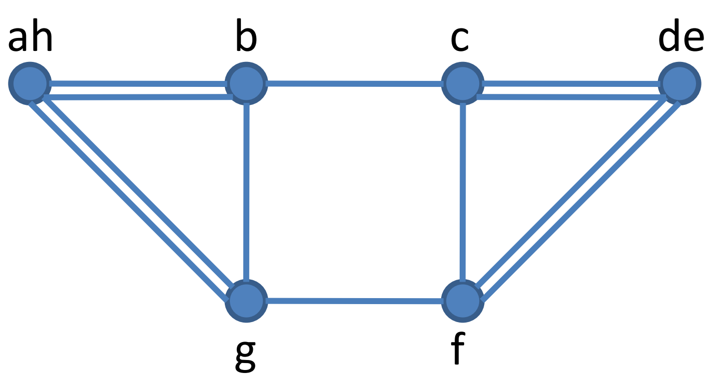
$i = 1$

1		14 edges to choose from. Pick $b - c$ with probability 1/14.
2		13 edges to choose from. Pick $g - f$ with probability 1/13.
3		12 edges to choose from. Pick $bc - gf$ with probability 2/12.

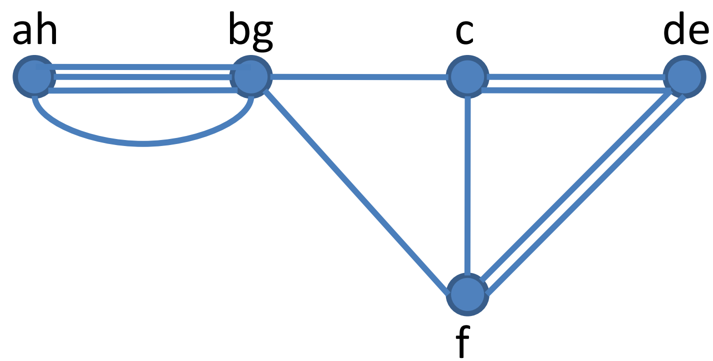
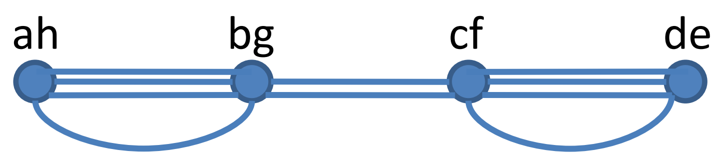
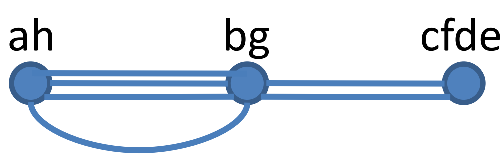
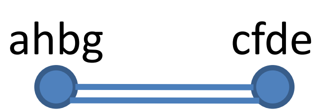
$i = 1$

4		10 edges to choose from. Pick $a - h$ with probability 1/10.
5		9 edges to choose from. Pick $d - e$ with probability 1/9.
6		8 edges to choose from. Pick $bcgf - de$ with probability 4/8.
7		Done. Min cut is 4 with probability 0.5. Vertex sets are $\{a, h\}$ and $\{b, c, d, e, f, g\}$

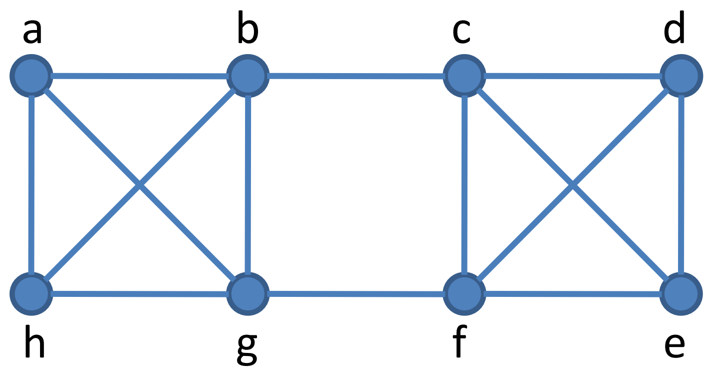
$i = 2$

<table>
<tr><td>1</td><td>
</td><td>14 edges to choose from. Pick $d-e$ with probability 1/14.</td></tr>
<tr><td>2</td><td>
</td><td>13 edges to choose from. Pick $a-h$ with probability 1/13.</td></tr>
<tr><td>3</td><td>
</td><td>12 edges to choose from. Pick $b-g$ with probability 1/12.</td></tr>
</table>

$i = 2$

4		11 edges to choose from. Pick $c - f$ with probability 1/11.
5		10 edges to choose from. Pick $cf - de$ with probability 4/10.
6		6 edges to choose from. Pick $ah - bg$ with probability 4/6.
7		Done. Min cut is 2 with probability 0.67. Vertex sets are $\{a, b, g, h\}$ and $\{c, d, e, f\}$

Contd…



i	C_i (# of edges)	Probability	Vertex Sets
1	4	0.5	$\{a, h\}$ $\{b, c, d, e, f, g\}$
2	2	0.67	$\{a, b, g, h\}$ $\{c, d, e, f\}$