Physical Design HW1

Problem 1 [Collaborator: None]

Given resolution R ; resolution constant k ; illumination wave length λ ; numeral apeture NA ; $NA = nsin(\theta)$; reflaction index n (water: 1.44, air: 1.00) ; $\theta < 68^o$

a.

From formula $R = k_1\lambda/NA$, we get

$R_{water} \geq 0.25 \times 193nm/1.44sin(68^o) = 36.14nm$

$R_{air} \geq 0.25 \times 193nm/1.00sin(68^o) = 52.04nm$

b.

immersion ($\times 1.2$) $157nm \times 0.25/1.44sin(68^o) = 29.40nm$

optical lithography ($\times 1.8$) $157nm \times 0.25/1.00sin(68^o) = 42.33nm$

EUV ($\times 3.2$) $13.5nm \times 0.25/1.00sin(68^o) = 3.64nm$

double ($\times 1.4$)

triple ($\times 1.9$)

quadruple ($\times 2.7$)

octuple ($\times 5.6$)

Therefore,

For 16nm, minimum cost is $1.2 \times 1.8 \times 1.4 = 3.024$, which is optical lithography + double patterning technology + immersion lithography

For 10nm, minimum cost is 3.2, which is EUV

For 5nm, minimum cost is 3.2, which is EUV

Problem 2 [Collaborator: None]

a.

Pseudo Code:

```
Input:
G(V,E)       // a tree graph
k            // independent set size parameter
A[|V|][2]=-1 // An array with size |A|$\times{}$2 with initial value -1
max_size(i,b){
    If(A[i][b]!=-1) Return A[i][b];
    Else{
        If(b==0){
            A[i][0]= Sum(max{max_size(j,0),max_size(j,1)}) on all childs;
            Return A[i][0];
        }
        If(b==1){
            A[i][0]= Sum(max{max_size(j,0)}) on all childs;
            Return A[i][1];
        }
    }
}
If(max{max_size(root_index,0),T(root_index,1)>=k})
    return True;
Else
    return False;
```

Complexity Analysis:
It is a dynamic programming algorithm. In each recursize function call, it has complexity with the order of number of it's child. Since each edge adn vertex are traversed only constant time in teh whole process, the time complexity of the whole algorithm is O(|V|+|E|). Moreover, since the order of vertex and edges are the same in a tree graph, it's complexity can be further express by O(|V|).
b.
Discussion:
The whole graph only contains 3type of subgrpah
(1) Independent vertex: if a vertex is of degree 0.
(2) A cycle graph: For a connected component in the graph, if it contains a loop, it must be a cycle graph; otherwise, there must be a vertex that has more than 2 degree, causing a contradition.
(3) A single chain graph: For a connected component in the graph, if it does not contains any loop, it must be a single chain graph. Otherwise, there must be a vertex tha has more than 2 degree, causing a contradiction.

If a connected compomemnt K_α is of type (1): then ,its max-ISP is 1
If a connected compomemnt K_α is of type (2): then ,its max-ISP is $\lfloor |K_\alpha|/2 \rfloor$
If a connected compomemnt K_α is of type (3): then ,its max-ISP is $\lfloor (|K_\alpha|+1)/2 \rfloor$

```
function traverse(Vi){
    Vi.mark = 1;
    if(Vi.next_node_1==NULL) // traverse done, forms single chain
        return false;
    else if(Vi.next_node_1.mark==0) // keep traversing
        return traverse(Vi.next_node_1)
    else if(Vi.next_node_1.mark==1){
        if(Vi.next_node_2==NULL) //traverse done, forms a single chain
            return false;
        else if(Vi.next_node_2.mark==1) // traverse collision, forms a cyle graph
            return true;
        else if(Vi.next_node_2.mark==0) // keep traversing
            return traverse(Vi.next_node_2)
    }
    Global cnt += 1;
}
main:
    max_ISP=0;
    for each Vi in G(V,E){
        if(Vi.mark==1) continue;
        if(Vi.next_node_1==NULL && Vi.next_node_12==NULL){
            // independent point
            max_ISP += 1;
            Vi.mark = 1;
            continue;
        }
        Global cnt = 1;
        if(Vi.next_node_1==NULL){
            traverse(Vi.next_node_2);
            max_ISP += floor((cnt+1)/2);
            continue;
        }
        if(Vi.next_node_2==NULL){
            traverse(Vi.next_node_1);
            max_ISP += floor((cnt+1)/2);
            continue;
        }
        collision = traverse(Vi.next_node_1);
        if(collision){
            // cycle graph
            max_ISP += floor(cnt/2);
            continue;
        }
        else{
            traverse(Vi.next_node_2);
        }
        max_ISP += floor((cnt+1)/2); // single chain
    }
    If(max_ISP>=k) return true;
    Else return false;
```

Complexity analysis:
Each vertex with be marked only once and the for loop traverse all vertex once. Therefore, O(|V|) complexity.

c.

[Theorem 1(Konig's Theorem). The cardinality of a maximum matching problem is equal to that of a minimum vertex cover on a bipartite graph]

Given a bipartite graph G(V,E), let M be all vertex that is matched, and L be the left set and R be the right set, and U be the set of unmatched vertex in L. Let Z contains all the vertex in U and all vertex on alternating path set begin from any vertex in U(An alternating path is a path that goes through match and unmatch edge alternatively).

First, it is obviously that we can’t find any vertex cover smaller than |M|, since we can’t even cover the match edges by less than |M| vertex.
Therefore, if we can construct a |M| size vertex cover, then it is the minimum vertex cover.
Let K=(L\Z)∪(R∩Z).
We can show that for any edge, either its left endpoint is in K or its right endpoint is in K. For any edge, if it is in the alternating path set, then its right endpoint must be in K by definition. For any edge, if it is not in the alternating path set and the edge is matched, then its left endpoint will not be in an alternating path set, so the left endpoint must be in K.For any edge, if it is not in tge alternating path and it is unmatch, then the left endpoint will not be in the alternating path set, too, so the left endpoint will be in the set K.
Since for any edge either one of its endpoint is in K, K forms a vertex cover.
Now, we only need to show that the cardinality of K is exactly |M|. We prove this by showing that all vertex in K is matched.
For any left endpoint, it must be matched, since L\Z excludes all unmatched left vertex. For any right endpoint, if it is unmatched, since it is on an alternating path, this alternating path must begin with and end with unmatched edges. If we change any match to unmatch and unmatch to match on this alternating path, then we can get a greater matching cardinaltiy, causing a contradiction. Therefore, the vertex in R∩Z must be matched vertex. Moreover, for any match edge, it can’t have both endpoint in K, so the cardinality of set K is |M|.
[Theorem 2. The minimum vertex cover problem is complement to the maximum independent set problem for any graph]
It is obviously that for any vertex cover, its complement is an independent set. If its complement is not an independent set, then there exists one edge that both of its endpoint is chosen. These two points will not be chosen on its complement graph, the edge is not covered, thus not a vertex cover.
Therefore, the complemnet of minimum vertex cover is a maximum independent set.
Algorithm:
Run any maxflow algorithm on the bipartite graph and get the maximum mathcing |M|. Then the k-ISP problem is true if |V|-|M| is greater than k.
Complexity analysis:
Assume that we choose Edmonds-Karp algorithm for solving max flow. Our complexity will be O(VE^2)

d.
Method for 3SAT reduction to n-ISP.
Suppose there is a 3SAT $(x_{1,1}+x_{1,2}+x_{1,3})(x_{2,1}+x_{2,2}+x_{2,3})$...$(x_{n,1}+x_{n,2}+x_{n,3})$.
Then we can form each CNF-term to a 3-cycle graph and get n 3-cycle graph(This can be done in linear time). Also, for any variable, we conenct them with its own inverse(this can be done in quadratic time). This process can be done in polynomial time.
We can show that if the 3SAT is satisfiable, the n-ISP problem is true.
If it is satisfiable, it means that we can find at least one ‘1’ in each CNF-term, choose one of them in each CNF-term forms a n-ISP solution.
If is is unsatisfiable, then, it means tha tthere exists at least one CNF-term that is all ‘0’ no matter how we assign our varaibles (notice that the 0 CNF-term will not be the same for different assignment). Since each CNF-term forms a 3-cycle graph, we can one pick one of them to form ISP. Therefore, if the 3SAT is unsatisfiable, it implies that the answer to the n-ISP problem is false.
Therefore, a 3SAT problem can be reduce to a n-ISP problem by our method.

Problem 3 [Collaborators: None]

a.
The weight W is stord in binary format with k bits, so W is in order O(2^k). Therefore, the O(nW) algorithm is actually a O($n2^k$) algorithm, which is exponential to input size k. It is not a polynomial time algorithm.
b.
The iteration times r is not determines, it can be really large in some case. Therefore, this algorithm is not a polynomial time algorithm.
c.
The variables $E,\ V,\ lg\,C$ are all input size. Therefore, this $O(EV\sqrt{lg\,C})$ algorithm is a polynomial time algorithm.

Problem 4 [Collaborators: None]

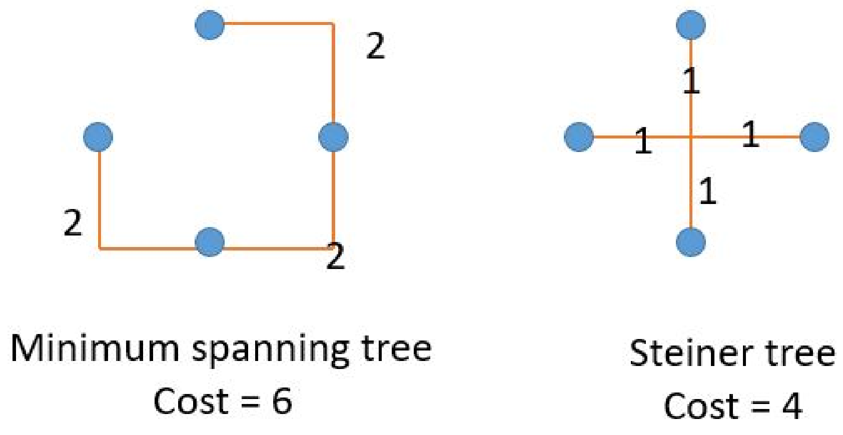
a.
If the optimal required number of bins is k and $k \leq \lceil S \rceil - 1$. Then, it can packed total $\lceil S \rceil - 1$ weight at max, which is smaller than S, causing a contradiction.
Therefore optimal number of bins required is at least $\lceil S \rceil$
b.
If there are more than two bin that are less than half-full, that means when we are running the FFD algorithm, there is one step that contains two bins that are less than half-full, but only one bin that is less than half-full in the prrvious step. If this happens, the new add object might be add to the previous bin that is less than half-full rather than adding a new bin, causing a contradiction.
Therefore, at most one bin is less than half-full.
c.
If the number of bins has $k \geq \lceil 2S \rceil + 1$. According to **b.**, at most one bin is less than half-full, we get:
$Total\ weight > (k-1)/2\ \geq\ \lceil 2S \rceil/2\ >\ 2S/2 = S$
Contradiction!!!
Therefore, number of bins is no more than $\lceil 2S \rceil$
d.
The optimal solution is $\lceil S \rceil$ shown in **a.**
The worst solution of FFD is $\lceil 2S \rceil$ shown in **c.**
Therefore approximate ratio is $\lceil 2S \rceil / \lceil S \rceil \leq 2$
e.
Pseudo code:

```
vector A = [];
for i = 1 to n { // traverse all objects
    find_bin = false;
    for j = 0 to A.size()-1{ // traverse all bins
        if(A[j]>=Si){ // put the object into bin j
            A[j] -= Si;
            find_bin = true;
            break;
        }
        if(find_bin == false) // add new bin
            A.push_back(1-Si);
    }
}
number of bins = A.size();
```

Complexity analysis:
The inner for loop has complexity O(i), so the total complexity is O(n^2)

Problem 5 [Collaborators: None]

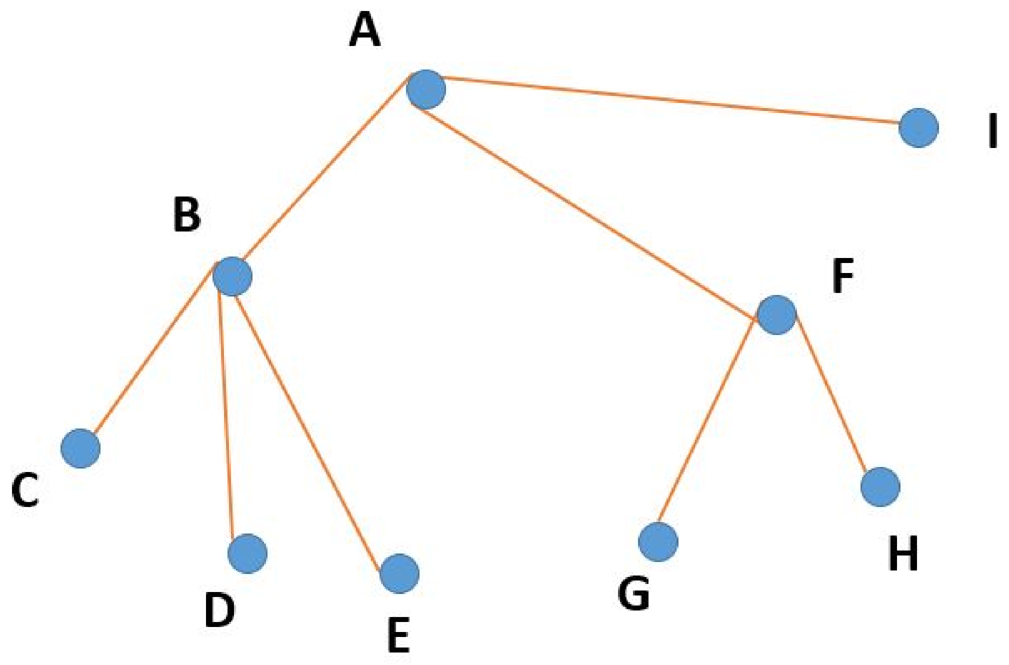
a.



b.
(Ref. Thm6. in lecture notes of Optimisation Algorithms
https://www.comp.nus.edu.sg/~stevenha/cs4234/lectures/03b.SteinerTree.pdf
(https://www.comp.nus.edu.sg/~stevenha/cs4234/lectures/03b.SteinerTree.pdf))

Definition of DFS-cycle graph and DFS-span tree:
Given a Steiner tree. We can traverse it by DFS order and forms a DFS-cycle graph C. We accomplish this by performing a DFS of the tree T, adding each edge to the cycle as it is traversed (both down and up). Notice that the cycle begins and ends at the root of the tree, and each edge appears in the cycle exactly twice. Also, we can form a DFS span tree T based on DFS order.
For example, the DFS-cycle graph for the following figure is
$(A,B)(B,C)(C,B)(B,D)(D,B)(B,E)(E,B)(B,A)$
$(A,F)(F,G)(G,F)(F,H)(H,F)(F,A)(A,I)(I,A)$

and the DFS-span tree of the following figure is:
$(A, B)(B, C)(C, D)(D, E)(E, F)(F, G)(G, H)(H, I)$



(1) A DFS-cycle graph has twice cost of a steiner tree:
Since each edge is traverse twice in C, the cost of graph C has twice cost of steiner tree T.

(2) The cost of a DFS-cycle graph is greater than a DFS-span tree:
This can be easily shown by general triangular inequality (Notice that this problem is defined on a metric space, so it is suitable for us to apply it, in this problem take L1 as our metric space), that is for any set of points $x_1, x_2, \ldots, x_n$, we have $\sum_{i=1 \sim n-1} x_i x_{i+1} \geq x_1 x_n$. Since both C and T is in DFS order, if there is a traverse edge (X_1, X_n) in T then there must be a edge traverse order $(X_1, X_2)(X_2, X_3)\ldots(X_{n-1}, X_n)$ in C, and by triangular inequality, this path in T is greater than C. Thus, overall cost of C is greater then that of T.

Conclusion:
By **(1)** and **(2)**, we can see that a DFS-span tree has at most twice cost of a steiner tree.

Problem 6 [Collaborator: None]

The constraint of the size of object can be see as there are only two types of size 1 and 2 (otherwise the constraint will be violated).
Out Extended-FFD algorithm is choosing the max size object to put into the 2D-bin rather than choosing by index.
We will show the optimality of our Extended-FFD.
Assume we use k bins in our Extended-FFD algorithm.
First, if the last add bin contains object with size 1, that means all the bins except the last are full, otherwise, the size 1 object must be add in the non-full bin. Since all bins except the last is full, that means that we cannot find any method to pack by less than k bins.
Second, if the last add object contains object with size 2, that means that the first k-1 bins are all packed with object of size 2, since we choose the max size object to packed first in our algorithm. It is obvious that for only one type of size, the greedy algorithm can get the most dense packing solution; therefore, it is optimal.

Problem 7 [Collaborators: None]

	a	**b**	**c**	**d**	**e**	**f**
a	0	0	2	3	0	0
b	0	0	6	2	1	0
c	2	6	0	0	0	4
d	3	2	0	0	5	3
e	0	1	0	5	0	8
f	0	0	4	3	8	0

Iteration1.

Step1.
Da=3-2=1
Db=3-6=-3
Dc=4-8=-4
Dd=5-8=-3
De=1-13=-12
Df=4-11=-7

gad=1-3-2x3=-8
gae=1-12-2x0=-11
gaf=1-7-2x0=-6 (max)
gbd=-3-3-2x2=-10
gbe=-3-12-2x1=-17
gbf=-3-7-2x0=-10
gcd=-4-3-2x0=-7
gce=-4-12-2x0=-16
gcf=-4-7-2x4=-19

So, Swap a and f and lock them. Current partition {f,b,c}{d,e,a}

Step2.
Db'=-3+2(0-0)=-3
Dc'=-4+2(2-4)=-8
Dd'=-3+2(3-3)=-3
De'=-12+2(8-0)=4

gbd=-3-3-2x2=-10
gbe=-3+4-2x1=-1 (max)
gcd=-8-3-2x0=-11
gce=-8+4-2x0=-4

So, Swap b and e and lock them. Current partition {f,e,c}{d,b,a}

Step3.
Dc''=-8+2(6-0)=4
Dd''=-3+2(5-2)=3

gcd=4+3-2x0=7

So, Swap c and d and lock them. Current partition {f,e,d}{c,b,a}

Largest partial sum of iteration 1 is -6-1+7=0, Terminate!
So the solution of KL is {a,b,c}

The following shows all possible solution:
{a,b,c}{d,e,f}: 3+3+4=10 (optimal)
{a,b,d}{c,e,f}: 2+7+8=17
{a,b,e}{c,d,f}: 5+8+13=26
{a,b,f}{c,d,e}: 5+9+15=29
{a,c,d}{b,e,f}: 0+10+10=20
{a,c,e}{b,d,f}: 3+10+14=27
{a,c,f}{b,d,e}: 3+6+11=20
{a,d,e}{b,c,f}: 3+2+11=16
{a,d,f}{b,c,e}: 2+7+12=21
{a,e,f}{b,c,d}: 5+6+7=18
Therefore, in this example KL does not get the optimal solution.

Problem 8 [Collaborators: None]

Calculate balance condition:
W=2+4+6+1+3+5=21
$0.3 = 0.3(21) - 6 \leq |A| \leq 0.3(21) + 6 = 12.3$
Iteration1.

Step1.

	a		**b**		**c**		**d**		**e**		**g(i)**
	F	T	F	T	F	T	F	T	F	T	
C1	0	0									0
C2	0	0	0	0							0
C3			0	0	+	0	+	0			2 <–
C4			+	0							1
C5	+	0			+	0			0	-	1
C6							+	0	0	-	0

Maximum gain C3 can balanced 0.3<12-6<12.3
Move C3

Step2.

	a	**b**	**c**	**d**	**e**	**gold**	**gnew**
C1						0	0
C2		+				0	1 <–
C4		-				1	0
C5			- -			1	-1
C6				- -		0	-2

Net	F	T	F	T
b	2	1	1	2
c	1	1	0	2
d	1	1	0	2

Maximum gain C2 and balanced 0.3<6-4<12.3
Move C2

Step3.

	a	b	c	d	e	gold	gnew
C1	+					0	1
C4		-				0	-1 <–
C5	-					-1	-2
C6						-2	-2

Net	F	T	F	T
a	2	1	1	2
b	1	2	0	3

Maximum gain C1 but unbalanced, so choose second max C4
Move C4

Step4.

	a	b	c	d	e	gold	gnew
C1						1	1 <–
C5						-2	-2
C6						-2	-2

Net	F	T	F	T
b	2	1	1	2

Maimum gain C1 and balanced
Move C1

Step5.

	a	b	c	d	e	gold	gnew
C5	-					-2	-3
C6						-2	-2 <–

Net	F	T	F	T
a	1	2	0	3

Maimum gain C6 and balanced
Move C6

Step6.
Move C5

Maximum partial sum=2+1-1+1=3 at step2. and step4.
Choose step4. so the current partition is {C4}{C1,C2,C3,C5,C6}

Iteration2.

Step1.

	a		b		c		d		e		g(i)
	F	T	F	T	F	T	F	T	F	T	
C1	0	-									-1
C2	0	-	0	0							-1
C3			0	0	0	-	0	-			-2
C4			+	0							1 <−
C5	0	-			0	-			0	-	-3
C6							0	-	0	-	-2

Maximum gain C4 but unbalance, choose second max gain C1
Move C1

Step2.

	a	b	c	d	e	gold	gnew
C2	+					-1	0
C3						-2	-2
C4						+1	+1 <−
C5	+					-3	-2
C6						-2	-2

Net	F	T	F	T
a	3	0	2	1

Maximum gain C4 and balanced. Move C4

Step3.

	a	b	c	d	e	gold	gnew
C2		-				0	-1 <–
C3		-				-2	-3
C5						-2	-2
C6						-2	-2

Net	F	T	F	T
b	1	2	0	3

Maximum gain C2 and balanced. Move C2

Step4.

	a	b	c	d	e	gold	gnew
C3		+				-3	-2
C5	+					-2	-1 <–
C6						-2	-2

Net	F	T	F	T
a	2	1	1	2
b	3	0	2	1

Maimum gain C5 and balanced. Move C5

Step5.

	a	b	c	d	e	gold	gnew
C3			+		+	-2	0
C6			+		+	-2	0

Net	F	T	F	T
c	2	0	1	1
e	2	0	1	1

No more moving, either one will cause unbalanced

Maximum partial sum is -1+1=0<=0, so terminate

Therefore, the final solution of FM heuristic is {C4}{C1,C2,C3,C5,C6}

Problem 9 [Collaborators: None]

(Ref. Lecture notes of multilevel coarsening algorithms
http://limsk.ece.gatech.edu/book/slides/pdf/hmetis.pdf
(http://limsk.ece.gatech.edu/book/slides/pdf/hmetis.pdf))
n1={a,c}
n2={b,g,e}
n3={c,e,f}
n4={a,d,f}
n5={d,f,g}
n6={e,h}
n7={f,g,h}
a.(hyperedge coarsening)
First, sort the hyperedge in increasing size and in lexicographical order if the net size is the same: n1, n6, n4 n2, n3, n5, n7
Step1. cluster n1 in to {a,c}
Step2. cluster n6 into {e,h}
Step3. a is marked in n4, skipped
Step4. e is marked in n2, skipped
Step5. a is marked in n3, skipped
Step6. cluster n5 in to {d,f,g}
Step7. f and g is marked in n7, skipped
So, the final cluster generated by hyperedge coarsening is:
$C1 : \{a, c\}$
$C2 : \{e, h\}$
$C3 : \{d, f, g\}$
$C4 : \{b\}$
Total 4 clusters generated.

b.(modified hyperedge coarsening)
After hyperedge coarsening, the uncoarsening nodes are: b.
So we cluster b into a new cluster.
Since only one node is uncoarsened, the clustering result of modified hyperedge coarsening is the same as hyperedge coarsening:
$C1 : \{a, c\}$
$C2 : \{e, h\}$
$C3 : \{d, f, g\}$
$C4 : \{b\}$
Total 4 clusters generated.

Problem 10 [Collaborators: None]

weight(n1) = wire_length(n1 is cut) - wire_length(n1 is not cut) = (13+8) - (8-3) = 21 - 5 = 16
weight(n2) = wire_length(n2 is cut) - wire_length(n2 is not cut) = (9+8) - (8-7) = 17 - 1 = 16
Therefore, both partition {c}{d} and {d}{c} has same wire length cost 16 + 6 = 22, where 6 is the sum of distance from node a to the center of A and node b to the center of A.

Problem 11 [Collaborators: None]

Extended Bin-Packing problem:
Given n bins each with volumn V_i and cost C_i where $i = 1 \sim n$, and m objects with volumn O_i where $i = 1 \sim m$, also assume that for any two objects O_i, O_j they satisfy either

$O_i = 2^k \times O_j$, or $O_j = 2^k \times O_i$. Find the lowest cost for packing all the object in to the bins. If there are no any packing strategy, the cost is infinity.

Sample solution:
The pseudo code for solving this problem is as follows:

```
Sort n bins in increasing Ci/Vi order
Sort m objects in decreasing Oi order.
current_bin_index = 0
For i = 0 : m-1
    if Bin_size[current_bin_index] > Om:
        Bin_size[current_bin_index] -= Om
    else:
        if(current_bin_index == n): // no solutions
            return inf;
        else:
            current_bin_index += 1
            Bin_size[current_bin_index] -= Om
return (current_bin_index + 1);
```

Complexity analysis
Sort bins: $O(nlgn)$
Sort objects: $O(mlgm)$
For loop: $O(m)$
Total: $O(mlgm + nlgn)$

Correctness
Assume we have got a solution in our algorithm. First, we look at the last packed bin. If there is a 1 unit object in the bin, that means all the bins excepts the last is full(with size <1 to full, but this size can't be packed by any objects), also the all full-packed bins are the lowest cost bins, so our solution is optimal. If there is no 1 unit object in the bin, that means all the objects are $2^k(k > 1)$ unit. In this condition, that means all the bins except ths last is at most $2^k - 1$ to full, so our solution is optimal. We can recursively apply this method and show that out algorithm is optimal!

Generalize problem and solution
If the volumn of objects have no constraints, then our algorithm might not be optimal. In this condition, we can solve it by the ILP solver.
$min \sum_{\forall j}((\sum_{\forall i} w_i x_{ij}) * C_j)$
$subject\ to$
$\sum_{\forall i} w_i x_{ij} \leq V_j, j = 1 \sim n$
$\sum_{\forall j} x_{ij} = 1, i = 1 \sim m$
$\sum_{ij} x_{ij} = n$
$x_{ij} \in \{0, 1\}$