1.
$Let\ X_{ij} be\ the\ indicator\ variable\ for\ pair(i, j)$

$Such\ that\ if\ pair(i, j)\ is\ inversion\ pair\ X_{ij} = 1\ else\ 0$

$with\ definition\ of\ inversion,\ such\ that\ there\ exist\ a\ inverse\ pair\ of\ (i, j)\ to\ be\ (a, b)$
$such\ that\ X_{ab} = 0\ so\ the\ P[X_{ij} = 1] = 1/2$

$$E[X] = \sum_{i<j} 1P[X_{ij} = 1] = \sum_{i<j} 1/2$$

$As\ there\ are\ total\ C^n_2\ pair\ for\ A$

$So\ E[X] = (1/2)(C^n_2)$

$E[X] = \frac{n(n-1)}{4}$

2.
$In\ each\ recursion\ that\ i\ would\ be\ drawn\ from\ Random(1, k)$
$If\ i\ in\ S\ as\ all\ element\ in\ S\ is\ drawn\ from\ (1, k - 1),\ insert\ k\ in\ S\ otherwise\ insert\ i$
$This\ mechanism\ guaranteed\ that\ every\ level\ of\ recursion,\ the\ number\ insert\ in\ S\ would\ be\ distinct$
$With\ this\ precaution,\ every\ time\ the\ recursion\ returns\ a\ distinct\ subset\ S\ in\ C^n_m\ choice$

$So\ claim\ that\ Random\ Sample\ would\ return\ a\ distinct\ subset\ S\ in\ C^n_m\ choice\ which\ P[S] = 1/C^n_m$
$is\ defined\ to\ be\ uniformly\ distributed$
$In\ base\ case\ m = 0$
$it\ is\ trivially\ true\ return\ a\ empty\ set$
$for\ m = 1$
$Random\ Sample\ return\ S\ with\ one\ element\ chose\ from\ k = (n - m) choices = 1/k$
$The\ probability\ of\ returning\ a\ subset\ uniformly\ would\ be\ 1/C^k_m = \frac{m!(k-m)!}{k!}\ by\ m = 1, it\ becomes\ 1/k$
$so\ it\ is\ true$

$By\ assuming\ that\ the\ subset\ [1, n]\ drawn\ by\ Random\ Sample\ is\ uniformly\ at\ random$
$such\ that\ presented\ as\ 1/(C^n_m) = \frac{m!(n-m)!}{n!}$
$In\ next\ recursion\ with\ i = n + 1\ and\ j = m + 1$
$the\ distinct\ element\ is\ choose\ from\ n + 1\ choices\ is\ distinct$
$S\ is\ guaranteed\ to\ be\ a\ distinct\ subset\ of\ size\ m + 1\ indicate\ that\ the\ probability\ of\ choose\ this\ distinct\ subset$
$is\ 1/C^{n+1}_{m+1} = \frac{(m+1)!(n-m)!}{(n+1)!} = \frac{j!(i-j)!}{i!} = 1/C^i_j\ is\ also\ uniform$
$so,\ this\ implies\ if\ n, m\ is\ true\ that\ returning\ a\ subset\ of\ [1, n] in\ uniform\ way\ in\ Random\ Sample$
$n + 1\ is\ also\ true\ that\ returning\ a\ subset\ of\ [1, n + 1] with\ size\ m + 1\ in\ uniform\ way,\ thus$
$Random - Sample\ returns\ a\ subset\ of\ [1,\ n]\ of\ size\ m\ drawn\ uniformly\ at\ random$

3.

$a)$

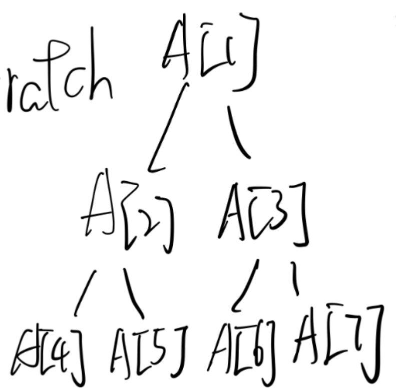
3. (10 points) Let $A[1..n]$ be an array of n distinct integers. This problem is about rearranging the elements of A so that it becomes an array representation of a binary min-heap. Assume that n is one less than a power of 2. Recall that A can be viewed as an array representation of a binary tree with $A[1]$ being the root at level 0. The children of the root is at level 1, and so on. For all $i \in [1, n]$, we use $\ell(i)$ to denote the level of $A[i]$. Let $L = \max\{\ell(i) : i \in [1, n]\}$. The height of $A[i]$ is defined as $L - \ell(i)$. So the root has height L and its children has height $L - 1$.

(a) Let h be any integer in the range $[0, L]$. Derive the indices of the elements of A that have height h. Derive the number of the elements of A that have height h.

(b) Consider an element $A[i]$. Suppose that the subtrees rooted at the children of $A[i]$ are already binary min-heaps. Describe how you can update the subtree rooted at $A[i]$ so that this subtree becomes a binary min-heap in time bounded by the height of $A[i]$.

(c) Use (b) to design a recursive algorithm that turns $A[1..n]$ into a binary min-heap. Explain the correctness of your algorithm. Show that the running time of your algorithm is $O(n)$.

a) Scratch



level	height
0	2
1	1
$L=2$	0

$$n = 2^{L+1} - 1$$

$$L = \log_2(n+1) - 1$$

$$h = L - \ell(i)$$

$$h = \log_2(n+1) - 1 - \ell(i)$$

since $\ell(i) = \lceil \log_2(i+1) \rceil - 1$

So $h = \log_2(n+1) - \lceil \log_2(i+1) \rceil$

$$\lceil \log_2(i+1) \rceil = \log_2(n+1) - h$$

$$i+1 = \left\lfloor \frac{2^{\log_2(n+1)}}{2^h} \right\rfloor$$

$$i = \left\lfloor \frac{n+1}{2^h} \right\rfloor - 1$$

Such that element of height h

$$i \in \left[\frac{n+1}{2^{h+1}}, \frac{n+1}{2^h} - 1 \right]$$

number of element from height 0 to height h
minus height 0 to height h+1

let K number of height 0 to height h

$K = 2^{L-h+1} - 1$ such that number of element hav
height h to be

$$2^{L-h+1} - 1 - (2^{L-(h+1)+1} - 1)$$

e

$$= 2^{L-h+1} - 2^{L-h}$$

$b)$ $compre\ the\ root\ A[i]\ with\ the\ childrens\ A[2i] and\ A[2i + 1]$
$if\ A[i]\ is\ already\ minimum\ compare\ to\ A[2i]\ and\ A[2i + 1]$
$then\ it\ already\ been\ a\ min\ heap\ rooted\ at\ A[i]$

$if\ A[i]\ is\ not\ the\ minimum,\ such$
$create\ a\ function\ update(node\ A[j])$
    $if\ no\ A[2i]\ and\ A[2i + 1]$
        $return$
    $let\ k = min(A[2i], A[2i + 1])$
    $then\ swap(A[i], k)$
    $update(k)$
$with\ this\ update\ function\ would\ only\ take\ in\ either\ A[2i]\ and\ A[2i + 1]\ subtree$
$if\ there\ are\ n\ nodes\ the\ running\ time\ for\ comparison\ would\ be\ T(n) = T(n/2) + 1$
$by\ master\ theorem,\ T(n) = log_2 n = O(log_2 n)$
$since\ root\ have\ height\ L\ as\ stated,\ L = log_2(n + 1) - 1 = O(log_2 n)$
$so\ is\ bounded\ by\ the\ height\ of\ A[i]$

$c)$

$function\ build(array\ A,\ number\ i)$
    $if\ i == 1$
        $return$
    $build(A, i - 1)$
    $do\ update(A, i)$
$initial\ call\ would\ be\ build(A, n)$

$this\ algorithm\ run\ in\ reverse\ direction\ of\ array\ A$
$the\ base\ case\ n == 1\ is\ trivially\ true$
$assume\ that\ for\ i - 1\ size\ of\ heap\ is\ already\ in\ min\ heap\ property$
$if\ there\ i\ elements\ such\ that\ A[i]\ is\ parent\ of\ A[i - 1]\ and\ A[i - 2]$
$with\ assumption\ subtree\ of\ root\ A[i - 1]\ and\ A[i - 2]\ is\ already\ in\ min\ heap\ property$
$so\ with\ the\ update\ function\ called$
$it\ will\ update\ the\ subtree\ rooted\ at\ A[i]\ so\ that\ this\ subtree\ becomes\ a\ binary\ min - heap$
$result\ in\ maintaining\ in\ a\ min\ heap\ property$
$so\ returning\ in\ size\ of\ i - 1\ min\ heap \Rightarrow the\ algorithm\ would\ generate\ size\ i\ min\ heap$
$so\ the\ algorithm\ is\ true$

$Function\ called\ would\ deep\ into\ update\ take\ the\ bound\ of\ the\ node's\ height\ h = \frac{n+1}{2^k} time$

$by\ induction\ each\ time\ the\ node\ at\ index\ i\ would\ only\ take\ O(log(\frac{n+1}{2^k}))\ as\ k\ is\ height\ of\ index\ i$

$so\ the\ total\ running\ time\ would\ be\ \sum_{k=0}^{log(n)} \frac{n+1}{2^k} * O(k) = (n + 1) \sum_{k=0}^{log(n)} \frac{k}{2^k}$

$= O(n + 1 \sum_{k=0}^{\infty} \frac{k}{2^k}),\ with\ geometric\ sequence\ \sum_{k=0}^{\infty} \frac{k}{2^k} = \frac{1/2}{(1-1/2)^2} = 2$

$so\ the\ running\ time\ would\ be\ O((n + 1) * 2) = O(n)$

4.

$let\ d\ be\ an\ array\ storing\ all\ the\ [s_j, e_j]\ of\ I_j\ with\ length\ n$

$d\ =\ sorted\ d\ with\ ascending\ order\ of\ s_j\ in\ each\ I_j$

$end\ =\ d[0][1]$

$S\ =\ [end],\ i\ =\ 0$

$For\ j\ -\!>\ 1\ to\ n\ -\ 1\#loop\ in\ the\ Intervals\ and\ start\ greedy$

$\quad If\ d[j][0]\ <\ end$

$\quad\quad end\ =\ min(end, d[j][1])$

$\quad\quad S[i]\ =\ end$

$\quad else$

$\quad\quad add\ d[j][1]\ in\ S$

$\quad\quad i\ =\ i\ +\ 1$

$\quad\quad end\ =\ d[j][1]$

$return\ S$

$Explanation:\ the\ algorithm\ start\ by\ sorting\ the\ array\ d\ such\ this\ give\ a\ standard\ to\ check$
$then\ sets\ the\ first\ optimal\ solution\ to\ be\ the\ e_0$

$running\ in\ a\ for\ loop,\ looping\ in\ each\ intervals\ start\ and\ end$
$check\ for\ I_j,\ if\ the\ s_j would\ be\ smaller\ than\ the\ I_{j-1}'s\ end$
$if\ it\ is,\ that\ means\ the\ minimum\ of\ e_j and\ e_{j-1} would\ definitely\ hit\ two\ intervals$
$otherwise, for\ the\ next\ interval\ as\ now\ s_{j+1} \geq s_j \geq s_{j-1} because\ the\ sorted\ property$
$so\ there\ must\ no\ element\ in\ I_{j+1}\ can\ hit\ I_{j-1}$
$with\ this\ say,\ the\ set\ need\ to\ add\ one\ more\ instance\ to\ make\ sure\ can\ hit\ all\ intervals$
$and\ we\ choose\ \ e_j\ of\ I_j,\ to\ be\ the\ new\ optimal\ solution\ for\ the\ next\ interval\ to\ check$

$for\ base\ case\ there\ only\ one\ Interval$
$the\ algorithm\ return\ S\ with\ start\ of\ Interval\ clearly\ hit\ the\ interval\ so\ is\ trivially\ true$

$assuming\ in\ number\ i\ intervals,\ S\ including\ the\ optimal\ sets\ of\ solution\ as\ each\ of\ element$
$hit\ I_a to\ I_b with\ s_a \leq s_b, a, b\ \leq\ i\ as\ shown\ in\ algorithm,\ now\ the\ end\ is\ denoted\ with\ e_k\ k\ \leq\ i$
$and\ it\ is\ lived\ in\ all\ intervals\ arbitrary\ I_a to I_b$
$now\ get\ into\ the\ next\ loop, denoted\ as\ I_{i+1}, check\ if\ s_{i+1} <\ \ e_k,\ if\ it\ is\ that\ means\ there\ must$
$an\ element\ live\ in\ I_a to\ I_b also\ live\ in\ I_{i+1},\ because\ the\ array\ is\ sorted$

$in\ the\ mean\ time,$

$and\ now\ check\ if\ e_{i+1} < e_k,\ if\ it\ is,\ means\ e_k is\ not\ live\ in\ I_{i+1}, with\ the\ greedy\ choice$

$as\ e_k hit\ I_a to I_b \Rightarrow e_{i+1} can\ also\ hit\ I_a to I_b\ also\ I_{i+1},\ so\ the\ e_k should\ be\ substituted\ by\ e_{i+1}$

$otherwise,\ e_{i+1} > e_k, since\ s_{i+1} < e_k \Rightarrow e_k also\ lived\ in\ I_{i+1}, but\ e_{i+1} not\ live\ in\ I_a to I_b$

$that\ means\ e_k is\ still\ the\ optimal\ solution\ to\ hit\ I_a to I_b and\ I_{i+1}$

$However,\ if\ s_{i+1} > e_k,\ it\ means\ there\ are\ no\ element\ in\ I_{i+1} would\ hit\ all\ intervals\ I_a to I_b$

$in\ this\ case,\ we\ have\ to\ add\ one\ more\ element\ in\ Set\ S\ to\ make\ sure\ all\ element\ in\ S\ can\ hit\ all\ I_a\ to\ I_b\ and\ I_{i+1}$

$thus,\ the\ result\ returning\ S\ is\ still\ be\ a\ smallest\ set\ of\ time\ instances\ that\ hit\ all\ i + 1\ intervals$

$so\ S\ is\ a\ smallest\ set\ of\ time\ instances\ hit\ i\ intervals$

$\Rightarrow S\ is\ still\ be\ a\ smallest\ set\ of\ time\ instances\ that\ hit\ all\ i + 1\ intervals$

$induction\ complete,\ the\ algorithm\ is\ true$

$With\ the\ array\ d\ of\ length\ n\ is\ sorted\ by\ ascending\ order\ of\ each\ s_j,\ such\ that\ it\ takes\ time\ O(nlog(n))$

$then\ the\ algorithm\ runs\ in\ each\ intervals\ and\ do\ constant\ time\ comparison\ = O(1)\ of\ n - 1\ times$

$such\ that\ = O(n)$

$So\ in\ total\ this\ algorithm\ runs\ in\ T(n) = O(nlog(n)) + O(n) \Rightarrow T(n) = O(nlog(n))\ as\ O(nlog(n))\ dominates$

5)

running time $T(n)$

$A_{\sigma(1)}$: $A_{\sigma(1)} = t_{\sigma(1)}$

$A_{\sigma(2)}$: $A_{\sigma(1)} + t_{\sigma(2)} = t_{\sigma(1)} + t_{\sigma(2)}$

$A_{\sigma(3)}$: $A_{\sigma(1)} + A_{\sigma(2)} + t_{\sigma(3)} = t_{\sigma(1)} + t_{\sigma(1)} + t_{\sigma(2)} + t_{\sigma(3)}$

$A_{\sigma(4)}$: $A_{\sigma(1)} + A_{\sigma(2)} + A_{\sigma(3)} + t_{\sigma(4)} = 3t_{\sigma(1)} + 2t_{\sigma(2)} + t_{\sigma(3)}$
$+ t_{\sigma(4)}$

so for running time $A_{\sigma(j)}$

$$= \sum_{i=1}^{j} (j-i) t_{\sigma(i)} + t_{\sigma(j)}$$

b

as total completion time is

$$\sum_{j=1}^{n} A_{6(j)} \text{ with each } A_{6(j)} = \sum_{i=1}^{j} (j-i) t_{6(i)} + t_{6(j)}$$

$$\text{so total} = \sum_{j=1}^{n} \sum_{i=1}^{j} (j-i) t_{6(i)} + t_{6(j)}$$

with inducted above, as i get smaller the jod require to do more repetitions of work for a single CPU machine do n job in order of $A_{6(1)} \dots A_{6(n)}$

as $t_{6(1)} < t_{6(2)} < \dots < t_{6(n)}$

would definitly minize the time as there are $k_{6(1)} \dots k_{6(n)}$

$$k_{6(1)} > k_{6(2)} > \dots k_{6(n)}$$

of $t_{6(j)}$ do $k_{6(j)}$'s repetition of work