Assignment 1

Ahiad Shaiker 308560671 , Nitzan Guetta 307937169

1. The functions by increasing order: $f_4(n) = \frac{1}{n}$, $f_1(n)$ =2017 ,

$$f_9(n) = \log(\sqrt{n}) , f_{11}(n) = \log(n^{10}) ,$$
$$f_{10}(n) = \log(2^n n^2) , f_2(n) = 2^{log_{\sqrt{2}} n},$$
$$f_{12}(n) = n^2 + \log(n) + n , f_3(n) = 2^{\sqrt{n}} , f_5(n) = 3^n,$$
$$f_7(n) = n^n , f_8(n) = 3^{2^n} , f_6(n) = 2^{3^n}$$

a) $\boldsymbol{\frac{1}{n} = O(2017)}$

$$so\ there\ are\ \mathrm{n > n0}, c > 0\ such\ that\ \frac{1}{n} \le c * (2017)$$
$$\rightarrow divide\ by\ 2017\text{:}\ \frac{1}{2017n} \le c$$
$$which\ there\ are\ for\ any\ n0, and\ c \ge 2$$

b) $\boldsymbol{2017 = O(\log(\sqrt{n}))}$

$$so\ there\ are\ \mathrm{n > n0}, c > 0\ such\ that\ 2017 \le c * \log(\sqrt{n})$$
$$\rightarrow divide\ by\ \log(\sqrt{n})\text{:}\ \frac{2017}{\log(\sqrt{n})} \le c$$
$$so\ when\ \log(\sqrt{n}) > 2017, c > 2$$
$$\log\left(n^{\frac{1}{2}}\right) > 2017 \rightarrow \frac{1}{2}\log(n) > 2017 \rightarrow \log(n) > 4034 \rightarrow n > 12$$
$$which\ there\ are\ for\ n0 \ge 12 , and\ c \ge 2$$

c) $\boldsymbol{\log(\sqrt{n}) = \Theta(\log(n^{10}))}$

$$so\ we\ need\ to\ prove\ that\ \log(\sqrt{n}) = \Theta(\log(n^{10}))$$

$$which\ means\ that\ there\ are\ \text{(for any)}\mathrm{n > n0}, c1, c2 > 0$$
$$such\ that\ c2 * \log(n^{10}) \le \log(\sqrt{n}) \le c1 * \log(n^{10})$$
$$c2 * \log(n^{10}) \le \log(\sqrt{n}) \rightarrow c2 * 10 * \log(n) \le \frac{1}{2}\log(\mathrm{n}) \rightarrow c2 * 10 \le \frac{1}{2}$$
$$\rightarrow c2 \le \frac{1}{20}$$
$$\log(\sqrt{n}) \le c1 * \log(n^{10}) \rightarrow \frac{1}{2}\log(\mathrm{n}) \le c1 * 10\log(n) \rightarrow \frac{1}{20} \le c1$$

Which exists for any n0, c1=c2=$\frac{1}{20}$

d) $\log(n^{10}) = O\left(\log(2^n n^2)\right)$

$$so\ there\ are\ \mathrm{n} > \mathrm{n0}\,, c > 0\ such\ that\ \log(n^{10}) \le c * \log(2^n n^2)$$

$$at\ first\ we'll\ simplify\ the\ expression\ \log(2^n n^2): \log(2^n n^2)$$

$$= \log(2^n) + \log(n^2) = \mathrm{n} * \log(2) + 2 * \log(n)$$

$$= n + 2\log(n)$$

$$so\ now\ we'll\ divide\ by\ n + 2\log(n): \frac{\log(n^{10})}{n + 2\log(n)} \le c \rightarrow$$

$$so\ for: \log(n^{10}) \le n + 2\log(n)\,, c > 1 \rightarrow 10\log(n) \le n + 2\log(n) \rightarrow$$

$$8\log(n) \le n, so\ for\ n0 = 256, c = 2$$

e) $\log(2^n n^2) = O\left(2^{\log_{\sqrt{2}} n}\right)$

$$so\ there\ are\ \mathrm{n} > \mathrm{n0}\,, c > 0\ such\ that\ \log(2^n n^2) \le c * 2^{\log_{\sqrt{2}} n}$$

$$at\ first\ we'll\ simplify\ the\ expression\ \log_{\sqrt{2}} n: \log_{\sqrt{2}} n = \frac{\log_2 n}{\log_2 \sqrt{2}}$$

$$= \frac{\log(n)}{\frac{1}{2}} = 2\log(n) \rightarrow so: 2^{\log_{\sqrt{2}} n} = 2^{2\log(n)}$$

$$from\ (d)\ we\ know\ that: \log(2^n n^2) = n + 2\log(n)$$

$$we'll\ divide\ by\ 2^{2\log(n)}: \frac{n + 2\log(n)}{2^{2\log(n)}} \le c \rightarrow \frac{2\log(n)}{2^{2\log(n)}} + \frac{n}{2^{2\log(n)}} \le c$$

$$\boldsymbol{so\ when}\ 2\log(n) < 2^{2\log(n)}\ \boldsymbol{and}\ n < 2^{2\log(n)}, \boldsymbol{c > 2}$$

- $2\log(n) < 2^{2\log(n)} \rightarrow \log(2\log(n)) < \log\left(2^{2\log(n)}\right) \rightarrow$ $\log(2) + \log(n) < 2\log(n)\log(2) \rightarrow 1 + \log(n) < 2\log(n) \rightarrow 1$ $< \log(n)\,, which\ exists\ for\ n0 = 3$
- $n < 2^{2\log(n)} \rightarrow \log(n) < \log\left(2^{2\log(n)}\right) \rightarrow \log(n) <$ $2\log(n)\log(2) \rightarrow 0 < \log(n)\,, which$ exists $for\ any\ n0$

so: $for\ n0 = 3, c = 2$

f) $\boldsymbol{2^{\log_{\sqrt{2}} n} = O(n^2 + \log(n) + n)}$

$$so\ there\ are\ \mathrm{n} > \mathrm{n0}\,, c > 0\ such\ that: 2^{\log_{\sqrt{2}} n} \le c *\ n^2 + \log(n) + n$$

$$from\ (e)\ we\ know\ that\ 2^{\log_{\sqrt{2}} n} = 2^{2\log(n)}$$

$$we'll\ divide\ by\ n^2 + \log(n) + n: \frac{2^{2\log(n)}}{n^2 + \log(n) + n} \le c$$

$$\boldsymbol{so\ when}: \frac{2^{2\log(n)}}{n^2 + \log(n) + n} < 1\,, \boldsymbol{c > 2}$$

$$\rightarrow 2^{2\log(n)} < n^2 + \log(n) + n \rightarrow \log\left(2^{2\log(n)}\right) < \log(n^2 + \log(n) + n)$$

$$\rightarrow 2\log(n)\log(2) < \log(n^2 + \log(n) + n)$$

$$\rightarrow \log(4)\log(n) < \log(n^2 + \log(n) + n) \rightarrow 4n < n^2 + \log(n) + n$$

$$\rightarrow 0 < n^2 + \log(n) - 3n \rightarrow 0 < n + \frac{\log(n)}{n} - 3 \rightarrow 3 - \frac{\log(n)}{n} < n\,,$$

$$which\ exists\ for\ n0 = 4 \rightarrow c = 2$$

g) $\boldsymbol{n^2 + \log(n) + n = O\left(2^{\sqrt{n}}\right)}$

$so\ there\ are\ \ \mathrm{n} > \mathrm{n0}\ , c > 0\ such\ that: n^2 + \log(n) + n \leq c * 2^{\sqrt{n}}$

$divide\ by\ 2^{\sqrt{n}}: \frac{n^2+\log(n)+n}{2^{\sqrt{n}}} \leq c \rightarrow\ \frac{n^2}{2^{\sqrt{n}}} + \frac{\log(n)}{2^{\sqrt{n}}} + \frac{n}{2^{\sqrt{n}}} \leq c$

$$\boldsymbol{so\ when}: \frac{n^2}{2^{\sqrt{n}}} < 1\ \boldsymbol{and}\ \frac{\log(n)}{2^{\sqrt{n}}} < 1\ \boldsymbol{and}\ \frac{n}{2^{\sqrt{n}}} < 1\ , \boldsymbol{c > 3}$$

- $\frac{n^2}{2^{\sqrt{n}}} < 1 \rightarrow\ n^2 < 2^{\sqrt{n}} \rightarrow \log(n^2) < \log\left(2^{\sqrt{n}}\right) \rightarrow 2log(n) <$ $\sqrt{n}\log(2) \rightarrow \qquad 2log(n) < \sqrt{n},\ which\ exists\ for\ n0 = 32^2$
- $\frac{\log(n)}{2^{\sqrt{n}}} < 1 \rightarrow \log(n) < 2^{\sqrt{n}} \rightarrow \log(\log(n)) < \log\left(2^{\sqrt{n}}\right) \rightarrow$

$\log(\log(n)) < \sqrt{n}\log(2) \rightarrow \log(\log(n)) < \sqrt{n}, which\ exists\ for\ n0 = 1$

- $\frac{n}{2^{\sqrt{n}}} < 1 \rightarrow n < 2^{\sqrt{n}}, which\ exists\ for\ n0 = 8$

$$so, n0 = 32^2, c = 3$$

h) $\boldsymbol{2^{\sqrt{n}} = O(3^n)}$

$$so\ there\ are\ \ \mathrm{n} > \mathrm{n0}\ , c > 0\ such\ that: 2^{\sqrt{n}} \leq c * 3^n$$

$$divide\ by\ 3^n: \frac{2^{\sqrt{n}}}{3^n} \leq c \rightarrow\ \left(\frac{2}{3^{\sqrt{n}}}\right)^{\sqrt{n}} \leq c\ , when: \left(\frac{2}{3^{\sqrt{n}}}\right) < 1, c > 2$$

$$so:\ \left(\frac{2}{3^{\sqrt{n}}}\right) < 1 \rightarrow 2 < 3^{\sqrt{n}}\ which\ exists\ for\ n0 = 1 \rightarrow\ \ c = 2$$

i) $\boldsymbol{3^n = O(n^n)}$

$$so\ there\ are\ \ \mathrm{n} > \mathrm{n0}\ , c > 0\ such\ that: 3^n \leq c * n^n$$

$$divide\ by\ n^n\ :\ \frac{3^n}{n^n} \leq c \rightarrow \left(\frac{3}{n}\right)^n \leq c, when: \left(\frac{3}{n}\right) < 1, c > 2$$

$$so\ :\ \left(\frac{3}{n}\right) < 1 \rightarrow 3 < n, n0 = 4 \rightarrow c = 2$$

j) $\boldsymbol{n^n = O\left(3^{2^n}\right)}$

$$so\ there\ are\ \ \mathrm{n} > \mathrm{n0}\ , c > 0\ such\ that: n^n \leq c * 3^{2^n}$$

$$divide\ by\ 3^{2^n}: \frac{n^n}{3^{2^n}} \leq c, when\ \frac{n^n}{3^{2^n}} < 1, c > 2$$

$$so: \frac{n^n}{3^{2^n}} < 1 \rightarrow\ \ n^n < 3^{2^n} \rightarrow \log(n^n) < \log\left(3^{2^n}\right) \rightarrow nlog(n) < 2^n\log(3)$$

$$\rightarrow \frac{nlog(n)}{\log(3)} < 2^n \rightarrow \log\left(\frac{nlog(n)}{\log(3)}\right) < n\log(2)$$

$$\rightarrow \log\left(\frac{n}{\log(3)}\right) + \log\left(\frac{log(n)}{\log(3)}\right) < n$$

$$\rightarrow \log(n) + \log\left(log(n)\right) - 2\log(\log(3)) < n$$

$$\rightarrow\ \ , which\ exists\ for\ any\ n0 \rightarrow c = 2$$

k) $\boldsymbol{3^{2^n} = O(2^{3^n})}$

$$so\ there\ are\ \mathrm{n} > \mathrm{n0}\ , c > 0\ such\ that: 3^{2^n} \le c * 2^{3^n}$$

$$divide\ by\ 2^{3^n}: \frac{3^{2^n}}{2^{3^n}} \le c, when \frac{3^{2^n}}{2^{3^n}} < 1, c > 2$$

$$so: \frac{3^{2^n}}{2^{3^n}} < 1 \to\ 3^{2^n} < 2^{3^n} \to \log\left(3^{2^n}\right) < \log\left(2^{3^n}\right) \to 2^n \log(3) < 3^n \log(2)$$

$$\to\ 2^n \log(3) < 3^n \to \log(3) < \frac{3^n}{2^n} \to\ \ \log(3) < (\frac{3}{2})^n \to \log(log3)$$

$$< \log\left(\left(\frac{3}{2}\right)^n\right) \to \log(log3) < nlog\left(\frac{3}{2}\right) \to \frac{\log(\log(3))}{log\left(\frac{3}{2}\right)} < n$$

$$\to \frac{\log(\log(3))}{log(3) - \log(2)} < n\ , for\ n0 = 1 \to c = 2$$

2. Answers:

a) True, let's say that $f(n) = n^n$.

We'll check if $f(n-k) \neq \theta(f(n))$.

So, $f(n-k) = (n-k)^{n-k},\ we'll\ check\ if\ there\ are$

$\mathrm{n} > \mathrm{n0}\,, c1, c2 > 0\ such\ that\ c1 * (n^n) \leq (n-k)^{n-k} \leq c2 * (n^n).$

$$\lim_{n\to\infty} \frac{(n-k)^{n-k}}{(n^n)} = \lim_{n\to\infty} \frac{\frac{(n-k)^n}{(n-k)^k}}{(n^n)} = \lim_{n\to\infty} \frac{(n-k)^n}{(n^n)(n-k)^k}$$

$$= \lim_{n\to\infty} \frac{1}{(n-k)^k} \frac{(n-k)^n}{(n^n)} =$$

$\left[\lim_{x\to\infty}[f(x)g(x)] = \lim_{x\to\infty}[f(x)] * \lim_{x\to\infty}[g(x)]\right] = \lim_{n\to\infty} \frac{1}{(n-k)^k} *$
$\lim_{n\to\infty} \frac{(n-k)^n}{(n^n)}$ = 0* $\lim_{n\to\infty} (\frac{n-k}{n})^n$ = 0 * $\lim_{n\to\infty} (1 - \frac{1}{\frac{n}{k}})^n = \quad 0 *$
$\lim_{n\to\infty} \ ((1 - \frac{1}{\frac{n}{k}})^{\frac{n}{k}})^k = \ 0 * e^k = \ 0$

➔ $(n-k)^{n-k} \leq c2 * (n^n).divide\ by\ n^n\ we'll\ get$

$\frac{(n-k)^{n-k}}{(n^n)} \leq c2$, we see that $\forall \mathrm{n0}, c2 = 1\ the\ occasion\ is\ right.$

➔ $c1 * (n^n) \leq (n-k)^{n-k}.divide\ by\ n^n\ we'll\ get$

$c1 \leq \frac{(n-k)^{n-k}}{(n^n)}$, we see that $\forall \mathrm{n0}, c1 \leq$

$0\ the\ occasion\ is\ contradiction\ to\ the\ fact\ that\ c1 > 0$

b) False,

Let's falsely assume that there exist n0,c ≥0 such that ∀n>n0,

$(f(n))^2$≤c*$f(n)$

$therefore$, ∀n>n0, $f(n)$≤ c in contradiction to the fact that

f(n)= $\Omega(\log n)$

c) True,

Let $f(n), g(n)$ functions such that $f(n), g(n) \geq 1$.

We want to prove that $f(n) + g(n) = O\big(f(n) * g(n)\big)$.

So, we need to prove that $f(n) + g \leq c * f(n) * g(n)$.

$f(n), g \geq 1.\ so\ if\ we\ divide\ by\ \big(f(n) * g(n)\big),$

$\frac{1}{g(n)} + \frac{1}{f(n)} \leq c$ it's always positive because f(n),g(n)≥1.

$we\ choose\ c \geq 2\ and\ than\ for\ every\ n0 \leq n, the$ claim will be.

3. Answers:

a) Iteration method:

$T(n) = T(n^{\frac{1}{2}}) + 1 = \left[T\left(n^{\frac{1}{4}}\right) + 1\right] + 1 = T\left(n^{\frac{1}{4}}\right) + 2$

$= \left[T\left(n^{\frac{1}{8}}\right) + 1\right] + 2 = T\left(n^{\frac{1}{8}}\right) + 3$

…

$= \left[T\left(n^{\frac{1}{2^i}}\right) + 1\right] + i - 1 = T\left(n^{\frac{1}{2^i}}\right) + i$

After $* \; i = loglogn$ iterations we have reached $T(2)$.

$T(n) = T\left(\frac{1}{n^{2^{loglogn}}}\right) + loglogn = T(2) + loglogn$

$= \theta(1) + loglogn$

$* \; n^{\frac{1}{2^i}} = 2 \leftrightarrow \frac{1}{2^i} logn = 1 \leftrightarrow logn = 2^i \leftrightarrow i = loglogn$

b) Master method:

$5T\left(\frac{n}{2}\right) + n^3 logn$

$a = 5, b = 2, f(n) = n^3 logn$

$we\ will\ check\ now\ if\ there\ is\ \varepsilon > 0\ such\ that\ n^3 logn = \Omega\left(n^{log5+\varepsilon}\right).$

$let's\ check\ if\ there\ is\ \varepsilon > 0\ such\ that\ there\ is\ c > 0, n0 > 0,$

$0 \leq c * n^{log5+\varepsilon} \leq n^3 logn$

We will divide by $n^{log5+\varepsilon}$

$0 \leq c \leq \frac{n^3 \log n}{n^{log5+\varepsilon}}$

$\varepsilon = 0.001, c = 1, n0 = 100.$

Now we'll check if there is 0<c<1 such that ∀n≥n0,

$5\left(\frac{n}{2}\right)^3 log\frac{n}{2} \leq c * n^3 logn$

Let's divide by $n^3 logn$ and then

$\frac{\frac{5}{8} log\frac{n}{2}}{logn} \leq c$

$\frac{5}{8} * \log_n \frac{n}{2} \leq c$

C=0.99, n0=2.5

Therefore, by master method, $T(n) = \theta(n^3 logn)$

c) Substitution method:

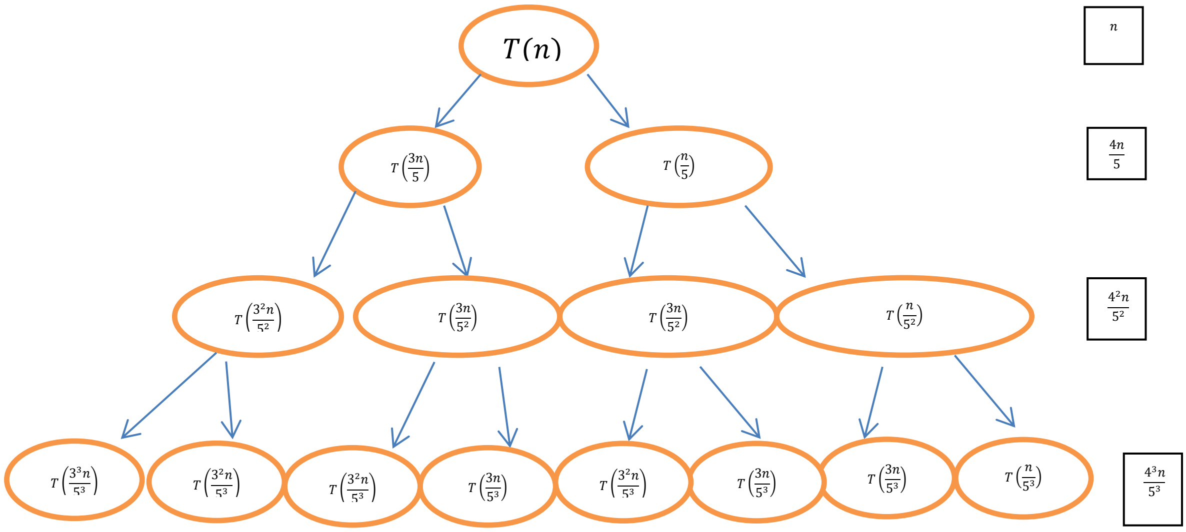
Out guess is that T(n)= $\theta(n)$.

Recurrence T(n) = (1 if n = 1 , T(cn)+T((1-c)n)+1 if n > 1 .

Base For n = 1, 1+1+1 = 3 = T(1).

Induction step Assume T(m) = θ(n) for all m < n.
We need to prove that T(n) = $\theta(n)$.
T(n) = T(cn)+T((1-c)n)+1 = cn+((1-c)n) + 1 = cn+n-cn+1 = n+1 = $\theta(n)$

d) We solved this by recurrence tree:
For this case recursion tree:



$$\frac{1}{\log 5} * nlogn = n\log_5 n \leq T(n) \leq n\log_{\frac{5}{4}} n = \frac{1}{\log\frac{5}{4}} * nlogn$$

➔ $T(n) \in \ \Theta(n\ log\ n)$

e) Iteration method:
$T(n) = 2T(n-1) + 1$ = $2[2T(n-2)+1]+1 = 4T(n-2)+3$
$= 4[2T(n-3)+1]+3 = 8T(n-3)+7 =$
...
$After\ i = n-1\ iterations\ we\ have\ reached\ T(1).$
$T(n) = \ 2^{n-1} * T(1) + (2^{n-1}-1) = \theta(2^n)$

4. Answers:

a) **<u>Function</u> BubbleSort(A[1…n])**

```
for i <-1 to n -1
        for j <- n downto i+1
                if A[j-1] > A[j]
                        temp <-A[j-1]
                        A[j-1] <- A[j]
                        A[j] <- temp
```

Line #num	cost	times
1	C1	n-1
2	C2	$\frac{n(n-1)}{2}$
3	C3	$\frac{n(n-1)}{2}$
4	C4	$\frac{n(n-1)}{2}$
5	C5	$\frac{n(n-1)}{2}$
6	C6	$\frac{n(n-1)}{2}$

So:

$$T(n) = (n-1) * c1 + \frac{n(n-1)}{2} * c2 + \frac{n(n-1)}{2} * c3 + \frac{n(n-1)}{2} * c4 + \frac{n(n-1)}{2} * c5 + \frac{n(n-1)}{2} * c6 = (n-1) * c1 + \frac{n(n-1)}{2} * (c2 + c3 + c4 + c5 + c6) = \theta(n^2)$$

Because there are 2 for loops which the first one goes n times (for n variables), and the second runs each time a different amount of times (for different amount of variables) – from 1 to (n-1). Which means the second for loop runs like arithmetic series: $\frac{n(n-1)}{2}$, there for BubbleSort function is $\theta(n^2)$.

b) **<u>function</u> exp2(base , power)**

```
if (power = 0)
        return 1
else if (power = 1)
        return base
else return base * exp(base, power-1)
```

Line #num	cost	times
1	C1	n
2	C2	1
3	C3	n
4	C4	1
5	C5	n-1

So: $T(n) = n * c1 + 1 * c2 + n * c3 + 1 * c4 + (n-1) * 5 = n * c1 + c2 + n * c3 + c4 + n * c5 - c5 = n(c1 + c3 + c5) + c2 + c4 - c5 = \theta(n)$

The function starts with power=n, until power is down to 1. when power=1 the recursive function stops.

So we can also define the function by the rule:

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + 1 & n > 0 \end{cases}$$

That by iteration we can also see that:

$$T(n) = \mathrm{T}(n-1) + 1 = T(n-2) + 1 + 1 = \cdots .. = T(1) + n - 1 = n$$

There for $T(n) = \theta(n)$

c) **exp2**(base,power) is like:

```
function exp2(base , power)
        if (power = 0)
                return 1
        else if (power = 1)
                return base
        else if (mod(power, 2) = 0)
                tmp exp2(base, power/2)
                return tmp * tmp
        else
                tmp  exp2(base, (power-1)/2)
                return base * tmp * tmp
```

At the worst case, in every call of the recursive method we will call to exp2(base,power/2), therefore,

$T(n) = T\left(\frac{n}{2}\right) + 1$ and the solution will be $\theta(logn)$.

At the best case, in every call we will call to **Exp2**(base,(power-1)/2)

in this case,

$$T(n) = T\left(\frac{n}{2} - \frac{1}{2}\right) + 1$$

We will solve by iteration method:

$$T(n) = T\left(\frac{n}{2} - \frac{1}{2}\right) + 1 = T\left(\frac{n}{4} - \frac{1}{4} - \frac{1}{2}\right) + 2 * 1 =$$
$$T\left(\frac{n}{8} - \frac{1}{8} - \frac{1}{4} - \frac{1}{2}\right) + 3 * 1 =$$

$$T(\frac{n}{2^i} - \sum_{k=1}^{i} \frac{1}{2^k}) + i * 1 = T\left(\frac{n}{2^i} + \frac{i}{2^i} - 1\right) + i * 1 = \log\left(\frac{n+1}{2}\right) + 1$$
$$= \log(n+1) - log2 + 1 = \theta(logn).$$

Therefore, at all cases the solution will be $\theta(logn)$.

5. Answers:

a) **Function find (A[1…n],x)** //finds a number in sorted array. Returns index/-1 (binary search function ans)

```
low<-1
if (A[0] = x)
        return 0
while (low<A.length AND A[low-1] <x)
        low<-low*2
if (low>A.length)
        return BinarySearch(A, low/2+1, length, x) // length- for not get an exception- out of bounds
return BinarySearch(A, (low/2)+1, low, x) // (low/2)+1 because we've already check that low/2<x
```

Function BinarySearch(A[1…n],low,high,x) //ordinary binary search function, returns index/-1

```
while(low ≤high)
        mid<-(low+high)/2
        if (A[mid] = x)
                return mid
        else if (a[mid]>x)
                high<-mid+1
        else
                low<-mid-1
return -1
```

the find function runs o(log(d)) times to find the range where x should be (d- the values in the array former to x location), when we find the range, the binary search function will run, in ordinary case it will take O(log(n)), but we gave it a range of d, and that's mean that BS function will run in O(log(d)). So the total running time is O(log(d))+ O(log(d))=2 O(log(d))= O(log(d))

b) **Function median(A[1..n], B[1..m])**

We have two sorted arrays A,B, and their medias: (variabls) medA,medB.
At the first step, the value of medA,medB, will be the medians of each array A,B.
If medA=medB – then we return one of them.
Else, one of them grater. Lets suppose medA<medB
So, the median of the merged array will be in one of this two subarrays: B[0..B.length/2],A[A.length/2..A.length].
Else, medA>medB.

So, the median of the merged array will be in one of this two subarrays: A[0..A.length/2],B[B.length/2..B.length].

We'll repeat this, recursively, till we'll get to arrays in size 2. Then, we'll merge the both arrays into AB array in size 4, and then we get the median in O(1) – AB[AB.length/2].

Accept of the step we will divide the arrays for 2 parts, and repeat the function recursively, all the other steps are O(1).

This step of dividing and calling again the function operates O(log(n)) – because like when we do binary search – we dividing the array for 2 parts, and continue the search in one of the parts. So $2^i = n \rightarrow \log(2^i) = \log(n) \rightarrow i * \log(2) = \log(n) \rightarrow i = \log(n)$ => O(log(n))